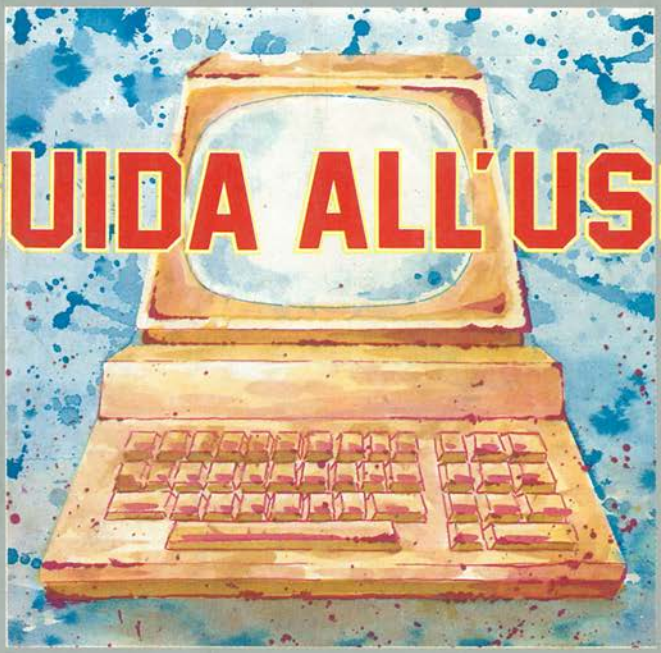
PET/CBM



GUIDA ALL'USO

VOL. 2

Adam Osborne
Carrol S. Donahue

EDIZIONE ITALIANA

GRUPPO
EDITORIALE
JACKSON

PET/CBM

GUIDA ALL'USO

VOL. 2

Adam Osborne
Carrol S. Donahue

EDIZIONE ITALIANA

GRUPPO
EDITORIALE
JACKSON

© Copyright per l'edizione originale McGraw-Hill Inc. 1980
© Copyright per l'edizione Italiana McGraw-Hill Inc.1984

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca Di Fiore, e l'Ing. Roberto Pancaldi.

Traduzione italiana a cura dell'Ing. Enrico Odetti.

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Fotocomposizione: Lineacomp S.r.l. - Via Rosellini, 12 - 20124 Milano

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

PREFAZIONE

Questo è il secondo volume della guida all'uso dei calcolatori CBM.

Esso contiene la continuazione di quanto iniziato a spiegare, nel primo volume, sui calcolatori CBM. In particolare il capitolo 6 descrive ampiamente e con altri dettagli le unità a cassette magnetiche, le unità a dischetti e le stampanti. Il capitolo 7 è dedicato alla descrizione del Sistema CBM; il capitolo 8 riporta in ordine alfabetico e con precisione formale tutte le istruzioni e le funzioni del BASIC CBM. Il volume termina con le appendici relative a tutta la guida.

SOMMARIO

CAPITOLO 6. UNITA' PERIFERICHE: UNITA' A CASSETTE MAGNETICHE, UNITA' A DISCHETTI E STAMPANTI 251

Memorizzazione di dati su una superficie magnetica 252
Il concetto di file. Trasferimento di dati da e verso una cassetta o un dischetto.

Gestione dei file su cassetta 261
Programmazione dei file di dati su cassetta. Formato dei file su cassetta.

File su dischetto 294
Registrazione dei dati su dischetto. Programmazione dei file su dischetto. Apertura dei file su dischetto. Chiusura di un file su dischetto. Errori di dischetto e parola di stato degli errori.

Operazioni di governo del dischetto 308
Preparazione e inizializzazione del dischetto. Riordino di un dischetto (Collect). Copia di file e dischetti. Cambiamento del nome di un file. Cancellazione di file.

File sequenziali di dati 318
Separatori di campo in file sequenziali. Scrittura di dati numerici in un file sequenziale. Scrittura di stringhe in un file sequenziale. Aggiunta di dati ad un file sequenziale. Fine del file (EOF).

File di dati relativi (BASIC 4.0) 331
Separatori di campo in un file relativo. Scrittura di dati numerici in un file relativo. Scrittura di stringhe in un file relativo. Posizionamento nei record di un file relativo.

Istruzione GET # per file su dischetto 337

File di programmi 339
Coda di lavori

Programmazione della stampante 342
Stampa di dati così come sono ricevuti. Stampa di dati con formato. Caratteri speciali di controllo della stampante. Impaginazione con formato. Definizione di caratteri personalizzati. Messaggi diagnostici della stampante.

CAPITOLO 7. INFORMAZIONI SUL SISTEMA CBM 365

Organizzazione del sistema CBM. Mappa della memoria. Memorizzazione delle istruzioni BASIC. Inizializzazione dell'area programmi di utente. Formato dei dati. Costanti. Rappresentazione dei caratteri. Programmazione in linguaggio Assembler. SYS. USR. File ad accesso diretto.

CAPITOLO 8. BASIC CBM 395

Modo immediato e modo differito. Revisioni del BASIC. Convenzioni sul formato.

Istruzioni BASIC 398
Funzioni 433
Funzioni di elaborazione dei testi dei calcolatori CBM 8000 445

Appendici
A. Codici dei caratteri CBM 449
B. Messaggi di errore 455
C. Tavole di conversione 467
D. Memoria ROM con revisione livello 2 477

INDICE ANALITICO 529

CAPITOLO 6

UNITA' PERIFERICHE: UNITA' A CASSETTE MEGNETICHE, UNITA' A DISCHETTI E STAMPANTI

Un sistema per elaborazione dati comprende, oltre al calcolatore stesso alla tastiera e al display video, anche delle unità periferiche per memorizzare programmi e dati. Per evitare di caricare un programma tramite la tastiera, ogni volta che si ha bisogno di eseguirlo, è possibile memorizzarlo invece su un dischetto "floppy" o su una cassetta magnetica. Come abbiamo già descritto nel capitolo 2 si può in seguito trasferire il programma in memoria centrale ed eseguirlo evitando così di caricarlo varie volte dalla tastiera.

Sui dischetti, come anche sulle cassette magnetiche, è possibile memorizzare sia programmi che dati. Consideriamo per esempio un programma per la gestione di una lista di indirizzi postali "mailing list". Il programma sarà memorizzato su una cassetta o su un dischetto e avrà il compito di gestire una lista di nomi e indirizzi. Anche questa lista sarà memorizzata sullo stesso supporto, cassetta o dischetto. Successivamente nomi e indirizzi saranno letti, dalla cassetta o dal dischetto, e saranno stampati come etichette della "mailing list". È ovvio che per questa ultima fase è richiesto l'uso di una stampante.

La maggior parte dei sistemi di calcolo comprende sempre una stampante per la stampa dei risultati, come le etichette della mailing list di cui abbiamo parlato. Una stampante è anche indispensabile se si vuole scrivere o solo modificare il testo di un programma. Infatti il modo più efficiente per cambiare o correggere un programma è quello di stamparne una "lista" e segnare a mano i cambiamenti che si vogliono fare sulla lista stessa e quindi caricare nel calcolatore i cambiamenti voluti.

In questo capitolo descriveremo la struttura dei programmi in BASIC necessari per gestire le unità a cassette, le unità a dischetto e le stampanti.

Nei calcolatori CBM si impiega il connettore IEEE 488, come dicevamo all'inizio, per collegare le unità a dischetto e le stampanti. Questo tipo di connessione è molto usato per collegare strumentazione elettronica od altre apparecchiature. Precisiamo che in questo libro non descriviamo lo standard IEEE 488 pur parlando di unità a dischetto e stampanti.*

---

* Per maggiori chiarimenti sul bus IEEE potete leggere "PET and the IEEE 488 Bus (GPIB)" di Eugene Fisher e C. W. Jensen edito da Osborn/McGraw-Hill, 1980.

MEMORIZZAZIONE DI DATI SU UNA SUPERFICIE MAGNETICA

IL CONCETTO DI FILE

Le informazioni sono memorizzate come archivi o "file" sulle cassette megnetiche o dischetti.

Per capire il concetto di file pensate ad una libreria: le cassette e i dischetti sono la libreria mentri i singoli libri sono i file.

Il concetto di file è molto semplice. Sia per leggere che per scrivere in un file è necessario prima "aprirlo" (in gergo tecnico "open") e poi alla fine "chiuderlo" ("close"). Questo equivale a prendere un libro dalla libreria e poi aprirlo, ma a differenza di un libro, vero e proprio, in un file si può sia scrivere che leggere. Quando un calcolatore scrive un programma o un testo di dati, su una cassetta o su un dischetto crea un nuovo file o ne estende uno vecchio.

Un file può avere una qualunque lunghezza con la sola limitazione della dimensione fisica del supporto magnetico utilizzato. Voi potete creare un nuovo file e non scrivere niente, cioè avere un file vuoto. Questo sarebbe come avere solo la copertina di un libro. Oppure potete avere un file che riempie completamente una cassetta o un dischetto e questo dipende solo dalla capacità massima della cassetta o del dischetto.

Teoricamente su una unica cassetta o dischetto potete avere al massimo 256 file diversi, ma in questo caso i singoli file saranno molto corti.

È importante precisare che la grandezza della memoria centrale del vostro calcolatore non è in rapporto con la lunghezza dei file di dati che potete creare. Un file di dati può essere infatti molto più lungo della memoria centrale del calcolatore. Una volta che un file di dati è "aperto" potete leggere un solo carattere come anche tanta informazione quanta ve ne può stare nella memoria centrale. Analogamente potete scrivere nei file pezzi successivi di testo che sulla cassetta o sul dischetto si accoderanno in sequenza.

File di programma

Ci sono due tipi di file: i file di programma e i file di dati. Un file di programma, come dice il nome stesso, contiene esattamente tutte le istruzioni di un programma.

Voi create un file di programma ogni volta che date il comando SAVE ad un programma caricato in memoria. Leggete invece un file di programma quando date il comando LOAD. Queste operazioni sono già state descritte nel capitolo 2.

Ogni file può avere un suo nome; il nome che assegnate ad un file di programma è anche il nome del programma. I nomi dei file nei calcolatori CBM possono essere lunghi fino a 128 caratteri, ma solo i primi 16 caratteri sono visualizzati. I nomi dei file su disco possono essere invece lunghi non più di 16 caratteri. È consigliabile quindi ridurre tutti i nomi di file a 16 caratteri al massimo.

A differnza dei file di dati, per un file di programma la grandezza della memoria centrale influenza la lunghezza massima di un singolo file. Questo perchè voi create un singolo file di programma quando date il comando SAVE. Quando invece caricate un programma in memoria voi dovete caricare tutto il programma e non solo una parte. Di conseguenza la massima lunghezza di un file di programma deve essere uguale o inferiore alla capacità di memoria disponibile nel vostro calcolatore CBM. Se avete un programma molto lungo, che non può essere caricato interamente nella memoria centrale, dovete ripartirlo in un certo numero di moduli ognuno di lunghezza giusta per essere caricato. Un modulo di programma alla volta sarà caricato e posto in esecuzione e in questo modo si otterrà l'esecuzione dell'intero programma. Successivamente in questo capitolo illustreremo la procedura esatta per eseguire in questa maniera programmi lunghi.

Uno dei vantaggi di operare con i file di programma consiste nel fatto che non è necessario conoscere la loro organizzazione interna. Quando "salvate" un programma su una cassetta o su un dischetto esso diviene subito un file che potrà essere successivamente ricaricato in memoria. Tutto quello che dovete sapere è come individuare il file di programma e per questo dovete identificarlo con un nome o con un indirizzo.

File di dati

Un file di dati, come è implicito nel nome, è un file che contiene informazioni che sono interpretate come dati e non come righe di programma. I file di dati sono creati, scritti e letti da programmi.

Record e campi

I file di dati sono suddivisi in "record" (un record è costituito da un gruppo completo di dati ed è l'elemento base di una registrazione). I record a loro volta sono ripartiti in "campi".

Un singolo campo contiene informazioni che possono essere rappresentate da una unica variabile. Di conseguenza un campo può contenere un numero intero, un numero con virgola o una stringa.

Un record contiene uno o più campi. Normalmente i record rappresentano informazioni ripetitive all'interno di un file.

Consideriamo l'esempio della mailing list. Tutta la mailing list è un file mentre ogni nome e indirizzo è un record. Se nomi e indirizzi sono caricati usando il programma del capitolo 5, allora ogni record contiene cinque campi: il campo del nome, il campo della strada, il campo della città, il campo dello stato e il campo del codice CAP. La organizzazione di questo file è illustrata in Figura 6-1.

Un file può contenere uno o più record e ogni record può contenere uno o più campi. **Il numero di record in un file e la massima lunghezza di un record variano con il tipo di file come vedremo nel seguito di questo capitolo. Tuttavia, per motivi pratici, la dimensione masima di un file è limitata solamente dalla capacità del dischetto.**

Nessuna limitazione è posta invece per la lunghezza dei record su cassetta magnetica. Un record può avere qualunque lunghezza che possa stare su una cassetta.

TRASFERIMENTO DI DATI DA E VERSO UNA CASSETTA O UN DISCHETTO

Chi operi per la prima volta con cassette o dischetti può cadere nell'errore di pensare che qualcosa non funzioni correttamente. Egli può istintivamente aspettar-

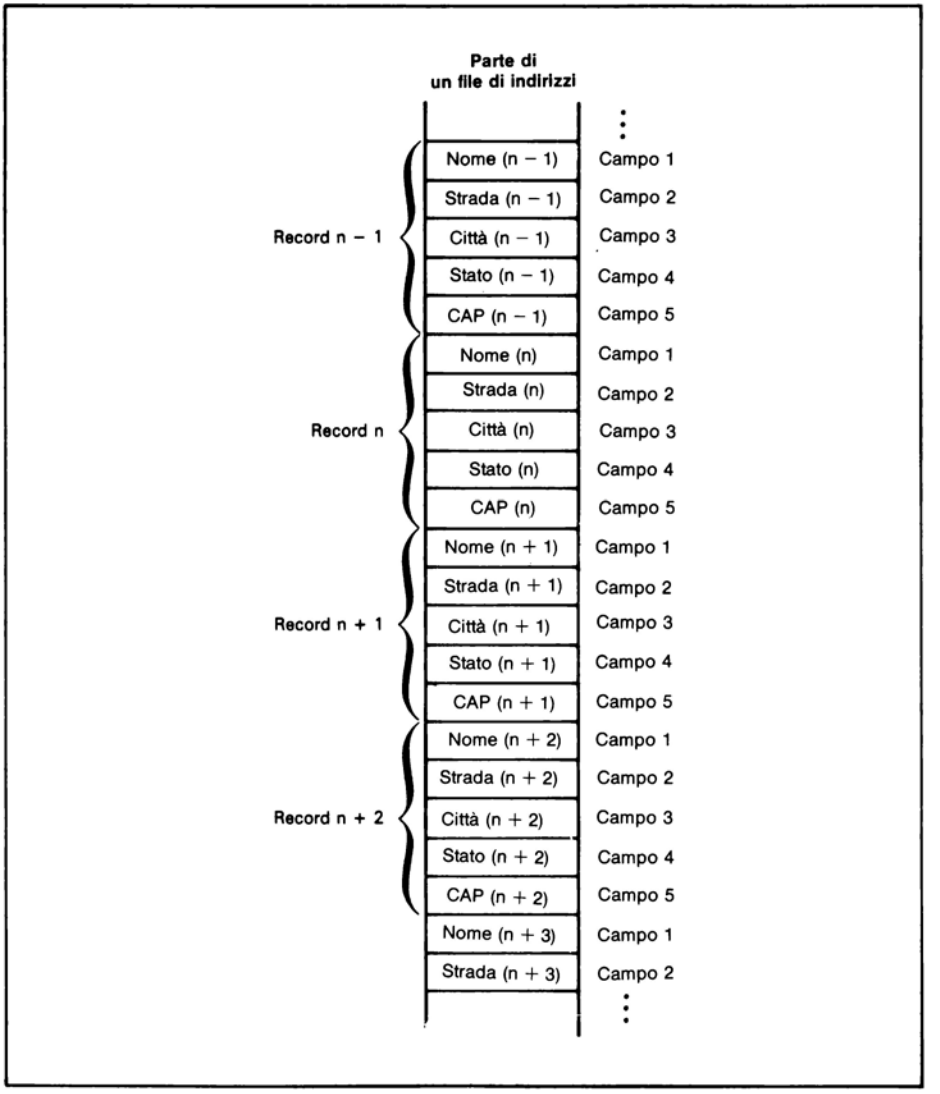


Figura 6-1: Illustrazione di quattro record di un file di indirizzi.

si che la cassetta o il disco si muova in risposta ad ogni comando di lettura o scrittura. L'unità a cassette dovrebbe sempre muovere la cassetta e l'unità a disco dovrebbe sempre attivare il dischetto. Alcune volte si nota infatti questo comportamento altre volte no. **Questo diverso comportamento è dovuto al fatto che il trasferimento dati tra il calcolatore e la periferica avviene tramite una piccola memoria che opera come un tampone (in inglese "buffer").**

Quando il calcolatore legge dal drive riempie dapprima il buffer e non ritorna a leggere dal drive sin tanto che i dati sono contenuti nel buffer.

I dati, prima di essere trasferiti sulla cassetta o sul dischetto, sono scritti nel buffer. Appena il buffer è riempito tutto il suo contenuto viene trasferito sulla memoria esterna e solo in questo momento si noterà una attività meccanica della periferica. Subito dopo non noterete nessun movimento sin che il buffer non sia nuovamente riempito.

Il buffer per le unità a cassette magnetiche è contenuto nella memoria del calcolatore CBM. È lungo 192 byte e può contenere 191 byte di dati. Invece il buffer per le unità a dischetti è contenuto nella periferica stessa e non nel calcolatore. Quest'ultimo buffer può contenere 156 caratteri. Ambedue questi buffer sono sufficientemente lunghi e quindi i drive saranno prevalentemente inattivi in attesa che il calcolatore riempia o vuoti il buffer corrispondente.

File logici e unità fisiche

Usiamo il termine "programmazione di ingresso/uscita" (o di "input/output") per descrivere la programmazione per il trasferimento di dati tra il calcolatore e le unità fisiche esterne. In questo caso le unità fisiche esterne sono le unità a cassette, le unità a disco e le stampanti.

Per effettuare una qualunque operazione di input/output la programmazione deve identificare l'unità fisica esterna di cui si richiede l'accesso.

La programmazione di ingresso/uscita è abbastanza complessa quando i dati sono trasferiti da o verso unità a cassette, a dischi o stampanti. È relativamente semplice invece nel caso di tastiere e display. Nel primo caso dovete aprire un "canale" tra il programma e l'unità fisica selezionata. Dovete poi, dopo aver effettuato le richieste operazioni di ingresso/uscita, chiudere il canale. Il linguaggio BASIC CBM riconosce i singoli canali mediante un numero compreso tra 0 e 255.

Un canale viene aperto mediante l'istruzione OPEN. I parametri dell'istruzione OPEN identificano l'unità fisica selezionata e il tipo di accesso, come illustrato nella Figura 6-2. Sino a che il canale non sia chiuso sarà sufficiente indicare il numero di canale, in ogni istruzione di ingresso o uscita, per descrivere completamente la natura di tali operazioni.

Ogni unità fisica ha il suo proprio ed unico numero fisico. Questo numero è usato come un parametro, nella fase di apertura del canale, in modo da identificare l'unità selezionata. Il numero di canale non ha invece una attribuzione permanente. **Questi**

numeri di canale sono spesso chiamati numeri di "file logico" o numeri di "unità logica".

Il nome "file logico" descrive molto bene il significato di un canale poichè un canale stabilisce un collegamento tra un programma e un file di dati.

I numeri dei file logici sono un concetto di programmazione. Come illustrato nella Figura 6-2 dovete iniziare ogni operazione di ingresso o uscita mediante una istruzione OPEN. Uno dei parametri di tale istruzione è appunto il numero di canale, o file logico; altri parametri identificano l'unità fisica e il modo in cui l'accesso si deve effettuare. **Dopo le istruzioni effettive di ingresso/uscita dovete dare un'istruzione finale di chiusura del canale: CLOSE.** Tale istruzione CLOSE richiede di indicare solo un parametro: il numero di canale o file logico. Le istruzioni OPEN e CLOSE sono collegate tra loro dal numero di file logico. Anche tutte le istruzioni di ingresso e di uscita fanno riferimento a tale numero di file logico sia per individuare la periferica a cui accedere sia per stabilire le modalità di accesso.

```
100 OPEN N, [Gli altri parametri identificano l'unità fisica
            su cui aprire il file e il tipo di accesso]
      •     N è il numero di file logico che può essere tra 0 e 255
      •
220 INPUT # N, [parametri]
      •
      •
      •
240 GET # N, [parametri]
      •
      •
      •
310 PRINT # N, [parametri]
      •
      •
      •
500 CLOSE N   Terminano le operazioni di ingresso/uscita iniziate
              con la OPEN
```

Tramite il valore N, che si riferisce alla OPEN della linea 100, si stabilisce l'unità fisica è il tipo di accesso in maniera univoca.

Figura 6-2: Illustrazione del concetto di numero di file logico

Il numero logico di file pone in relazione tra di loro le istruzioni OPEN, CLOSE, INPUT, GET e PRINT.

Una volta che un numero di file logico è stato posto in una istruzione OPEN non potete usarlo, come numero di un altro canale, sino a che non lo avete chiuso con una CLOSE. Se commettete questo errore il BASIC CBM vi darà un errore di sintassi. A parte quest'ultima restrizione non avete altra limitazione su come usare i numeri di file logico nel vostro programma.

Il numero di periferica identifica l'unità fisica che sarà selezionata dal calcolatore e appare come un parametro di una istruzione OPEN. Ogni unità fisica che può comunicare con un calcolatore CBM ha un suo numero fisso e permanente di periferica. Il calcolatore CBM, quando incontra un numero di periferica in una istruzione OPEN, attiva una parte della logica elettronica così da stabilire un collegamento fisico con l'unità identificata appunto da tale numero di periferica. **Nella Tabella 6-1 sono riportati i numeri di periferica riconosciuti dai calcolatori CBM.** Sebbene siano disponibili ben 256 numeri compresi tra 0 e 255, normalmente si usano solo i numeri da 0 a 30.

Tabella 6-1: Numero di periferica e indirizzi secondari corrispondenti

Periferica	Numero di periferica	Indirizzo secondario	Funzione operativa
Tastiera	0		
Cassetta # 1*	1 (imposto)	0 1	Apertura in lettura Apertura in scrittura
Cassetta # 2	2	2	Apertura in scrittura e aggiunta di un EOT (segno di fine nastro) alla chiusura
Display	3	nessuno	
Stampanti CBM 2022 e CBM 2023	4	0 1 2 3 4 5 6	Stampa di dati così come sono ricevuti Stampa di dati con un formato definito Riceve un formato da usarsi successivamente Riceve il numero di righe da stampare per pagina Abilita la stampa di messaggi diagnostici Crea un carattere speciale Stabilisce la spaziatura tra le righe (solo per CBM 2022)
Unità a disco (tutti i modelli)	8	0 1 2 — 14 15	Carica un programma nella memoria centrale Salva un programma su disco Indirizzi non usati Apertura del canale di comando
Altre periferiche collegate al bus IEEE 488	5,6,7,9		Vedere le istruzioni fornite dal costruttore della periferica
	da 31 a 255 non sono disponibili		

* Unità a cassetta montata nel calcolatore

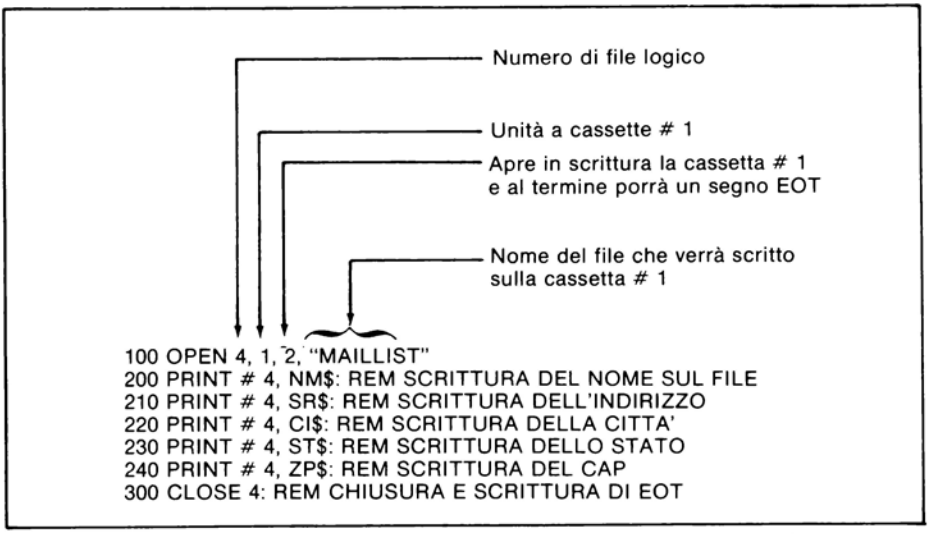


Figura 6-3: Uso dei parametri nelle istruzioni di ingresso/uscita

Molte unità fisiche rispondono, oltre al loro numero di periferica, anche ad altri numeri chiamati indirizzi "secondari". Per meglio comprendere il significato di questi indirizzi secondari è opportuno pensarli come dei "comandi", dati dal calcolatore alla periferica, per precisare quali operazione essa deve effettuare. **In Tabella 6-1 sono riportati gli indirizzi secondari delle unità fisiche più comunemente collegate ad un calcolatore CBM.** Non preoccupatevi per ora di studiare questi indirizzi secondari; avrete occasione di usarli frequentemente quando descriveremo in dettaglio la programmazione delle operazioni di ingresso e uscita.

Nella Figura 6-3 è illustrato in maniera completa l'uso dei vari parametri nelle istruzioni di ingresso/uscita.

Le cinque istruzioni PRINT #, riportate dalla riga 200 alla 240, scrivono le cinque parti di un nome e di un indirizzo di un archivio denominato MAILLIST e registrato su una cassetta inserita nel drive 1 dell'unità a cassette. Ad ogni istruzione PRINT # il calcolatore ricercherà il numero di file logico che appare dopo #. In Figura 6-3 questo numero è 4 per cui deve esserci una istruzione OPEN che specifica quali operazioni fare sul canale 4; come vedete questa OPEN è alla riga 100. Se mancasse la opportuna istruzione OPEN il calcolatore non eseguirebbe nessuna operazione di ingresso/uscita perchè non saprebbe che cosa fare. La nostra OPEN specifica il file logico o canale numero 4, l'unità fisica numero 1 (e cioè il registratore a cassette 1) e infine l'indirizzo secondario 2. Con questo indirizzo secondario è possibile scrivere nell'archivio, ma non è possibile leggerlo; inoltre, quando l'archivio sarà chiuso, verrà registrato un segno EOT (End Of Tape) di fine del nastro per impedire che altri dati vengano aggiunti all'archivio. Nella OPEN viene stabilito infine che il nome dell'archivio è MAILLIST.

Alla riga 300 è presente l'istruzione CLOSE che chiude il canale 4 per cui ogni cosa che era stata iniziata alla riga 100 dalla OPEN, viene terminata alla 300 dalla CLOSE.

Inoltre, siccome nell'istruzione OPEN della riga 100 era presente l'indirizzo secondario numero 2, quando sarà eseguita l'istruzione CLOSE verrà registrato sulla cassetta il segno speciale detto di fine nastro (EOT).

Così il file logico numero 4, che viene richiamato nelle istruzioni dalla riga 200 fino alla 300, pone in relazione tutte queste istruzioni con la OPEN della riga 100. Analogamente gli altri parametri della OPEN trovano il loro utilizzo nelle istruzioni successive.

Stato delle unità fisiche

Una stampante può ricevere dati dal calcolatore, ma ovviamente non può fornirli come periferica d'ingresso. Niente vieta però di eseguire una istruzione di INPUT che faccia riferimento ad un numero di file logico, inserito in una istruzione OPEN, che inizializzi una stampante in uscita.

Sebbene un registratore a cassette possa ricevere o trasmettere dati, l'indirizo secondario posto nella OPEN specificherà se l'operazione potrà essere solo di lettura o solo di scrittura. Può succedere quindi che voi erroneamente cerchiate di eseguire operazioni non permesse.

In altre parole, se cercate di eseguire le istruzioni PRINT, GET o INPUT in modo che la periferica sia chiamata a compiere una operazione per cui non è stata costruita o non è stata programmata, la periferica stessa vi avviserà con un errore di stato. Una unità fisica non svolgerà un compito che non le venga autorizzato da una OPEN; anche se fisicamente lo potrebbe svolgere. Per esempio se voi aprite una unità a cassette per una operazione di sola scrittura, eventuali istruzioni di INPUT o GET non saranno eseguite e daranno luogo ad un errore di stato.

Una unità fisica ritorna una informazione di stato a seguito di ogni istruzione di ingresso o uscita sia che le abbia eseguite con successo sia che non le abbia eseguite. L'informazione di stato è contenuta in 8 bit.

Per accedere allo stato basta riferirsi alla variabile ST. Per esempio l'istruzione:

```
10 X = ST
```

assegna il valore corrente dello stato, qualunque esso sia, alla variabile X.

In Tabella 6-2 sono riportati i valori dello stato generati dalle periferiche più comunemente collegate ai calcolatori CBM.

Quando in seguito scriverete programmi, che faranno riferimento alle varie unità fisiche, dovrete spesso impiegare questa Tabella 6-2.

Non cercate di controllare lo stato della tastiera e del display anche se essi hanno un loro numero di periferica.

Tabella 6: 2 Significati della parola (o byte) di stato ST

Periferica e operazione	**STATO**							
	00 00 001 (1)	**00 000 010 (2)**	**00 000 100 (4)**	**00 001 000 (8)**	**00 010 000 (16)**	**00 100 000 (32)**	**01 000 000 (64)**	**10 000 000 (–128)**
Lettura dalla Cassetta # 1 o # 2	Tutto regolare	Tutto regolare	Blocco corto Il blocco letto ha meno byte di quanti attesi	Blocco lungo Il blocco letto ha più byte di quanti attesi	Errore di lettura	Errore di checksum. Uno o più bit non sono stati letti correttamente	È stato incontrato un EOF	È stato incontrato un EOT
Verifica dalla Cassetta # 1 o # 2					Errore nell'operazione di VERIFY		Niente	
Unità a disco (tutti i modelli)	Perifica non disponibile in ricezione	Periferica non disponibile in trasmissione	Niente	Niente	Niente	Niente	EOF	Drive non presente
BUS IEEE 488	Superamento del tempo di ascolto	Superamento del tempo di chiamata					Fine della identificazione	Periferica non presente

Nella Tabella 6-2 sono riportati, per completezza, i valori di stato standard del "bus" IEEE 488 anche se in questo testo non si parla dell'interfacciamento tramite questo "bus".

GESTIONE DEI FILE SU CASSETTA

Descriveremo in questo paragrafo le procedure necessarie per creare, leggere e modificare file di dati su cassetta sotto il controllo dei programmi.

Alcune delle istruzioni BASIC per la gestione dei file, che useremo in questo capitolo, non sono ancora state illustrate per cui vi rimandiamo al capitolo 8 dove potrete eventualmente trovare una dettagliata spiegazione.

Per iniziare vi ricordiamo che potete programmare il calcolatore per leggere dati dalla cassetta oppure per scrivere dati su essa, ma non potrete mai comandare i soli movimenti meccanici. È importante sapere come funzionano i registratori per evitare che cerchiate di effettuare erroneamente operazioni non previste.

Su una cassetta i file sono memorizzati sequenzialmente. Una "intestazione" precede il primo file e un segno di fine nastro EOT (End Of Tape) segue l'ultimo file. Ogni file è terminato con un segno di fine file EOF (End Of File).

L'intestazione è scritta automaticamente all'inizio della cassetta quando vi scrivete sopra per la prima volta. In tal caso noterete che il registratore è in funzione senza che ve lo aspettiate, ma di questo non dovrete preoccuparvi.

Il calcolatore può ricercare un file mentre il nastro si muove in avanti sotto il comando PLAY, ma non può farlo con l'avanzamento veloce FAST FORWARD. Un segno EOF rivela la fine di un file mentre un segno EOT rivela la fine del nastro. Il byte di stato è pari a 64 nel caso di EOF mentre è pari a −128 nel caso di EOT.

I comandi di avvio del registratore devono essere dati manualmente, premendo gli appositi tasti, quando sono richiesti dal calcolatore. Non premete alcun tasto sul registratore prima che vi appaia il relativo messaggio sul display. Successivamente il calcolatore fermerà la cassetta al momento giusto e la farà ripartire automaticamente se dovrà fare una ulteriore operazione. Per questo motivo non toccate i tasti di vostra iniziativa.

Quando scrivete dei dati su una cassetta essa deve essere posizionata correttamente nel momento in cui inizia la registrazione, altrimenti i nuovi dati saranno sovrapposti alle vecchie registrazioni; se per caso la testina si trovasse sopra la parte iniziale trasparente del nastro non verrà fatta ovviamente alcuna memorizzazione. È buona norma, sia che abbiate una cassetta nuova o che ne usiate una vecchia, che la testina si trovi subito sul nastro magnetico vero e proprio.

Tenete presente ancora che la distanza tra un record o un file e il successivo è assicura automaticamente dal registratore stesso. Non create voi assolutamente dello spazio di separazione tra i record, o i file, facendo avanzare manualmente il nastro. Non potrete mai ri-registrare un record o un file sin tanto che la vostra capacità di riavvolgere il nastro, esattamente al punto voluto, non sia ottima.

Anche un piccolo errore porterà a delle registrazioni che non potranno più essere lette.

Quando volete leggere file di dati già registrati dovete prima assicurarvi che il nastro sia riavvolto ad un punto precedente a quello d'inizio del file che desiderate leggere. I calcolatori CBM possono trovare qualunque file dotato di norme con il normale avanzamento in avanti, ma non possono assolutamente fare una ricerca all'indietro riavvolgendo il nastro.

Non tentate mai di riscrivere una parte di un file già in precedenza registrato: è una operazione troppo rischiosa! Supponiamo per esempio di avere dieci nomi registrati su una cassetta e desideriamo cambiare il quinto nome.

In teoria potremmo leggere i primi quattro nomi e lasciare quindi posizionato il nastro all'inizio del quinto nome. A questo punto potremmo pensare di registrare il nuovo nome sopra al vecchio. In pratica è molto difficile che questo funzioni. La spiegazione sta nel fatto che i registratori a cassette non sono molto precisi meccanicamente. C'è quindi una elevata probabilità che si registri il nuovo dato un pò prima o un pò dopo del punto voluto così che il vecchio nome non sarebbe perfettamente cancellato.

Per aggiornare (in inglese "update") un file è necessario avere due drive. Da uno si legge il vecchio file e sull'altro si registra il nuovo con le modifiche. Attenzione dovete procedere in questa maniera anche quando avete da cambiare un solo dato tra centinaia!

Nel BASIC CBM non sono previste istruzioni che muovano o posizionino i nastri in un modo qualunque.

PROGRAMMAZIONE DEI FILE DI DATI SU CASSETTA

Un programma per accedere ad un file su cassetta deve prevedere questi tre punti:

1. OPEN: apertura del file
2. INPUT o GET o PRINT: operazioni di ingresso/uscita
3. CLOSE: chiusura del file.

Apertura di un file di dati su cassetta

La prima istruzione che dovete dare è la OPEN. Se la dimenticate avrete un errore di sintassi. Il formato dell'istruzione OPEN è il seguente:

OPEN N	Apre il file logico N. Seleziona il primo file incontrato sulla cassetta nel drive 1 e permette operazioni di lettura.
OPEN N, D	Apre il file logico N. seleziona il primo file incontrato sulla periferica D e permette operazioni di lettura. D deve essere pari a 1 per il drive 1 e 2 per il secondo drive.
OPEN N, D, S	Apre il file logico N. Seleziona il primo file incontrato sulla periferica D e permette le operazioni specificate dall'indirizzo secondario S (vedi Tabella 6-1). D deve essere pari a 1 per il drive 1 e 2 per il secondo drive.
OPEN N, D, S, nome	Apre il file logico N. Seleziona il file con il nome indicato sulla periferica D e permette le operazioni specificate dall'indirizzo secondario S (vedi Tabella 6-1).

L'istruzione OPEN può essere accompagnata da una combinazione di diversi parametri. N è l'unico parametro che deve essere sempre presente. D se manca è posto eguale a 1. S se manca è posto uguale a 0. Infine se manca il nome del file verrà selezionato il primo file incontrato in fase di lettura, oppure non verrà assegnato nessun nome in fase di scrittura.

Quando una istruzione OPEN è eseguita per aprire una unità a cassette in lettura, il calcolatore visualizzerà il seguente messaggio se nessun tasto del registratore è premuto:

```
PRESS PLAY ON TAPE #1
OK ←————————————— Un tasto del registratore è premuto; il nastro comincia a muoversi
```

Il calcolatore legge allora l'intestazione della cassetta. Se siamo in modo immediato appariranno i seguenti messaggi (le indicazioni tra parentesi saranno presenti solo se era stato dato il nome del file nell'istruzione OPEN):

SEARCHING [FOR nome]	Viene data la lista di tutti i file incontrati dal punto di partenza al file cercato
FOUND nome a	(di ogni nome vengono riportati solo i primi 16 caratteri)
FOUND nome b	
FOUND nome c	
FOUND nome d	
FOUND	Indicazione di file senza nome
FOUND	
FOUND [nome]	File ricercato
READY	File aperto

Nel caso di modo differito questo blocco di messaggi non appare.

Se la istruzione OPEN è data invece per aprire una cassetta in scrittura, il calcolatore CBM visualizzerà i seguenti messaggi se nessun tasto del registratore è premuto:

```
PRESS PLAY & RECORD ON TAPE #1
OK ←————————————————— Un tasto del registratore è premuto; il nastro comincia a muoversi
```

Il calcolatore CBM scrive quindi l'intestazione sulla cassetta e poi il nastro si ferma. Ecco alcuni esempi di istruzioni OPEN:

OPEN 1	Apre il file logico 1. Non essendo specificata alcuna unità fisica, viene scelta la cassetta # 1. Non è indicato alcun indirizzo secondario così è selezionata la fase di lettura (cioò l'indirizzo secondario 0). Giacchè non è specificato il nome del file, viene letto il primo file incontrato.
OPEN 1,1	Come sopra giacchè il secondo parametro equivale al valore di default.

OPEN 1,1,0	Ancora come sopra perchè il secondo e terzo parametro equivalgono al valore di default.
OPEN 1,1,0, "DAT"	Sempre come sopra, ma questa volta viene ricercato il file di nome DAT.
OPEN 3,1,2	Apre il file logico 3 sulla cassetta # 1. Scrive un nuovo file e pone alla fine un segno di "EOT". Il nuovo file non ha nome.
OPEN 3,1,2, "PENTAGRAM"	Come la OPEN precedente, ma con il nome PENTAGRAM dato al file.

Chiusura di un file di dati su cassetta

Le operazioni di apertura e chiusura di un file sono strettamente connesse per cui è opportuno descrivere adesso l'istruzione CLOSE prima di dire come si opera nel file. Ricordiamo però che l'istruzione CLOSE è sempre l'ultima di una sequenza di programmazione su file e dopo di essa non è più possibile accedere al file.

Il formato dell'istruzione CLOSE è il seguente:

```
CLOSE N
```

dove N è il numero del file logico o canale che appare come primo parametro nella corrispondente istruzione OPEN.

Se avete chiuso un file su cassetta dopo averlo letto non potete più continuare la lettura. Se però vi dimenticate la chiusura non commetterete nessun danno, ma farete solo della cattiva programmazione.

Nel caso di operazioni di scrittura è obbligatorio chiudere il file. Ricordiamo che i dati, prima di essere registrati sulla cassetta, sono depositati in un buffer di memoria. Quando il buffer è pieno viene trasferito automaticamente sulla cassetta. Alla fine della registrazione il buffer sarà molto probabilmente riempito solo in parte è sarà appunto il comando CLOSE a forzare il suo ultimo trasferimento. **Se per qualunque motivo il file non viene chiuso questo buffer residuo non viene registrato causando una evidente perdita di dati.** Oltre a questa funzione la chiusura del file fa registrare sulla cassetta un segno di fine file EOF che è assolutamente necessario per separare un file dal successivo. Senza il segno EOF il calcolatore leggerà il file successivo come se fosse la continuazione del precedente.

Dobbiamo precisare inoltre un altro aspetto della registrazione. Se all'apertura del file era indicato anche l'indirizzo secondario 2, è proprio alla chiusura che verrà registrato un segno di fine nastro EOT. L'EOT avvisa il calcolatore che sulla cassetta non sono presenti altri dati. Se esso manca il calcolatore può continuare a cercare altri file e leggere quindi vecchie registrazioni che noi presumiamo non siano più correttamente interpretabili.

È opportuno però precisare che la chiusura dei file di dati su cassetta può essere effettuata anche in altro modo. **Infatti quando si esegue l'istruzione END tutti i file su cassetta, ancora aperti, vengono automaticamente chiusi.**

E allora perchè chiudere separatamente ogni singolo file? Le ragioni sono due:

1. La chiusura separata vi obbliga a pensare in termini più logici e così saranno evitati tanti errori.

2. Possono essere aperti contemporaneamente non più di dieci file su cassetta.

Sono pochi i programmi che richiedono più di dieci file aperti nello stesso tempo. Tuttavia se non chiudete i file dopo averli usati, il vostro programma può terminare con troppi file aperti. Questa situazione può causare dei problemi, specialmente in programmi lunghi, che siano scritti in piccoli moduli. Se ogni modulo lascia aperti alcuni file è facile raggiungere il numero massimo di dieci file così che se aprite l'undicesimo commetterete un errore di esecuzione. E questo è uno dei peggiori errori da trovare e correggere. Perchè questo vostro programma può aver girato tante altre volte senza darvi errori.

Chiudere separatamente ogni file richiede poco impegno e può farvi risparmiare in seguito tanti errori durante l'esecuzione del programma.

L'istruzione CLOSE può essere eseguita sia in modo immediato che in modo differito.

Quando date l'istruzione CLOSE di un file aperto in scrittura e nessun tasto del vostro registratore è abbassato, il calcolatore vi richiederà di premere i tasti PLAY e RECORD:

```
PRESS PLAY & RECORD ON TAPE #1 <-- Premere i tasti
OK <------------------------------ Il nastro si muove per registrare il buffer
```

Se invece il file era aperto in lettura non dovrete premere nessun tasto.
Vi diamo alcuni esempi di istruzioni CLOSE:

```
 10 CLOSE  1          Chiude il file logico 1
 20 CLOSE 14          Chiude il file logico 14
210 A = 14
220 CLOSE A           Chiude il file logico 14
```

Come accedere a file di dati su cassetta

Una volta che un file di dati su cassetta sia aperto, potete scrivervi o leggerlo a seconda di quanto specificato nell'indirizzo secondario dell'istruzione OPEN. L'accesso al file può continuare sino a che non sia chiuso, ma ricordatevi che sia la lettura che la scrittura devono avvenire sequenzialmente. Il primo record, scritto o letto, deve essere sempre il primo record del file. Se desiderate per esempio leggere il decimo record del file, dovete prima leggere tutti i record dall'uno al nove. Analogamente per scrivere il decimo record dovete prima aver scritto gli altri nove.

Prima di effettuare una registrazione dovete assicurarvi di aver inserito le cassette giuste nei drive.

Se avete un solo drive vi consigliamo di inserire prima la cassetta con i programmi, caricare il programma desiderato, togliere questa cassetta e inserire quella per i dati. Solo alla fine di queste operazioni ponete in esecuzione il vostro programma. Se avete invece due drive potete tenere in uno la cassetta programmi e nell'altro la cassetta dati, sempre che il programma non richieda anche l'uso del primo drive per i dati.

Non abbassate i tasti dei drive prima che lo chieda il calcolatore.

Ricordatevi ancora che è compito dell'operatore posizionare correttamente le cassette magnetiche. Il calcolatore inizierà a scrivere o leggere la cassetta dal punto in cui si trova inizialmente il nastro e non è possibile accedere automaticamente a punti antecedenti.

Il comando per scrivere dati sulla cassetta è PRINT #:

PRINT # f, dati

dove:	f	è il numero di file logico definito nella OPEN e in CLOSE; può essere tra 1 e 255.
	dati	sono i dati da scrivere

Attenzione: il comando PRINT # non può essere dato nella forma compatta ? #. **PRINT # deve essere battuto per esteso.**

PRINT # trasferisce i dati nel buffer. Quando il buffer raggiunge la sua capacità di 191 byte totali viene trasferito sulla cassetta come un "blocco". Un blocco può quindi contenere parte di un record, un singolo record o più record.

Il comando PRINT # permette di registrare sia dati numerici che stringhe.

Scrittura di numeri su cassetta magnetica

Quando si scrivono dati numerici su una cassetta ogni numero è separato dal comando di ritorno del carrello (RETURN o CHR% (13)).

Proviamo a scrivere un programma, a cui diamo il nome NUM.PRINT #, per scrivere i numeri dall'1 al 10 su una cassetta.

Per prima cosa il programma visualizzerà i suoi compiti e darà le istruzioni di caricamento:

```
10 PRINT"◆◆ CREAZIONE DI DATI NUMERICI SU CASSETTA ◆◆":PRINT
20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆":PRINT
30 GET A$: IF A$="" THEN 30
```

La linea 20 avvisa l'utente di inserire una cassetta nel registratore, di riavvolgerla fino all'inizio e di premere RETURN quando è pronta. L'istruzione in linea 30 attende che un qualunque tasto sia premuto. Il calcolatore si pone così in ciclo di attesa per dar tempo di predisporre la cassetta.

Il tempo di ciclo programmato nella linea 30 ha però l'inconveniente che può essere terminato toccando qualunque tasto a differenza di quanto detto nel messaggio che avvisava di premere RETURN. È preferibile allora modificare come segue la linea 30:

```
30 GET A$:IF A$<>CHR$(13) THEN 30
```

Appena premuto RETURN il programma prosegue con l'istruzione OPEN di apertura del file su cassetta:

```
40 PRINT"◆◆ APERTURA DEL FILE ◆◆":OPEN 1,1,2,"NUMERI"
```

Questa istruzione OPEN apre il file logico # 1, seleziona l'unità fisica # 1 (cioè il registratore a cassette) e pone l'indirizzo secondario 2 (apertura solo per scrittura e segno EOT alla chiusura del file). Il nome del file di dati è NUMERI.

Successivamente con un ciclo FOR NEXT visualizziamo i numeri dall'1 al 10 e li registriamo sulla cassetta:

```
50 FOR N=1 TO 10
60 PRINT N          ←———— Visualizza N
70 PRINT#1,N        ←———— Scrive N sul file #1 (NUMERI)
80 NEXT N
```

PRINT N visualizza i numeri sul display. PRINT # 1,N li scrive invece sul nastro. Ricordate però che PRINT # non può essere sostituito da ? #. Esso deve essere scritto per esteso così come lo abbiamo scritto noi più sopra.

Errato	Corretto
?#1,N	PRINT#1,N
PRINT N	
PRINT #1,N	
PRINT#1N	
PRINT1,N	

I comandi errati daranno luogo a errori di sintassi, salvo PRINT N che visualizzerà la variabile N sul display.

Se tutto procede correttamente le linee dalla 50 alla 80 visualizzeranno i numeri e li scriveranno sul nastro:

Schermo

```
1
2
3
4
5
6
7
8
9
10
```

Rappresentazione dei dati sul nastro

L'istruzione PRINT# scrive un carattere di ritorno del carrello sulla cassetta così come l'istruzione PRINT visualizza un ritorno a capo. Così l'istruzione PRINT# della linea 70 scrive un ritorno del carrello dopo il numero N, proprio come PRINT, nella linea 60, da luogo ad un ritorno a capo dopo aver visualizzato N. **Vi consigliamo, per scrivere correttamente i numeri sulle cassette, di usare l'istruzione PRINT#** come usereste PRINT per scrivere una colonna verticale di numeri.

Dopo che tutti i dati sono stati scritti sul nastro bisogna chiudere il file con l'istruzione CLOSE.

```
90 PRINT"◆◆ CHIUSURA DEL FILE ◆◆": CLOSE 1
100 END
```

Fate attenzione di usare lo stesso numero di file logico sia nella OPEN che nella CLOSE.

```
OPEN   1,1,2, "NUMERI"
.
CLOSE 1
```

Il listato completo del programma NUM.PRINT # è il seguente:

```
10 PRINT"◆◆ CREAZIONE DI DATI NUMERICI SU CASSETTA ◆◆":PRINT
20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆":PRINT
30 GET A$: IF A$="" THEN 30
40 PRINT"◆◆ APERTURA DEL FILE ◆◆":OPEN 1,1,2,"NUMERI"
50 FOR N=1 TO 10
60 PRINT N
70 PRINT#1,N
80 NEXT N
90 PRINT"◆◆ CHIUSURA DEL FILE ◆◆": CLOSE 1
100 END
```

La sua esecuzione è:

```
◆◆ CREAZIONE DI DATI NUMERICI SU CASSETTA ◆◆

◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆

◆◆ APERTURA DEL FILE ◆◆

PRESS PLAY & RECORD ON TAPE #1
OK

 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 ◆◆ CHIUSURA DEL FILE ◆◆
```

Scrittura di stringhe su cassetta megnetica

Quando scrivete stringhe di variabili sulle cassette potete separare le singole variabili sia con una virgola che con un ritorno del carrello.

La funzione dei due separatori è però diversa. Infatti, quando leggerete dalla cassetta mediante l'istruzione INPUT # la lettura andrà avanti sino ad incontrare un carattere di ritorno del carrello. Di conseguenza se usate le virgole esse separano solamente le stringhe all'interno di un gruppo. **Un gruppo di stringhe viene sempre letto per intero ed ha ai suoi estremi i caratteri di ritorno del carrello.** Dopo l'ultima stringa dovete sempre registrare un ritorno carrello per farlo riconoscere poi dall'istruzione INPUT #.

Per separare con virgole le variabili di stringa è necessario conoscere delle tecniche particolari di programmazione. L'uso contemporaneo dei due separatori può dar luogo a confusione anche a programmatori BASIC molto esperti. Fate quindi molta attenzione agli esempi che seguono prima di scrivere voi stessi nuovi programmi.

Proviamo a modificare il programma NUM.PRINT # per scrivere invece le parole da "UNO" fino a "DIECI" come stringhe. Al nuovo programma diamo il nome PAROL.PRINT #. Le singole parole possono essere date sia con l'istruzione INPUT che con READ/DATA. Nel nostro esempio usiamo READ/DATA con la READ inserita nel ciclo FOR NEXT alla linea 60 e DATA posta alla nuova linea 110. Il programma e la sua esecuzione si presentano come segue:

PAROL.PRINT # 1

```
10 PRINT "◆◆ CREAZIONE DI UN FILE DI PAROLE ◆◆": PRINT
20 PRINT "◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆"
30 GET A$: IF A$="" THEN 30
40 PRINT "◆◆ APERTURA DEL FILE ◆◆": OPEN1,1,2,"PAROLNUM": PRINT
50 FOR N=1 TO 10
60 READ N$
70 PRINT N$
80 PRINT#1,N$
90 NEXT N
100 PRINT "◆◆ CHIUSURA DEL FILE ◆◆": CLOSE 1
110 DATA UNO,DUE,TRE,QUATTRO,CINQUE,SEI,SETTE,OTTO,NOVE,DIECI
120 END
```

```
◆◆ CREAZIONE DI UN FILE DI PAROLE ◆◆

◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆

◆◆ APERTURA DEL FILE ◆◆

PRESS PLAY & RECORD ON TAPE #1
OK

UNO
DUE
TRE
QUATTRO
CINQUE
SEI
SETTE
OTTO
NOVE
DIECI
◆◆ CHIUSURA DEL FILE ◆◆
```

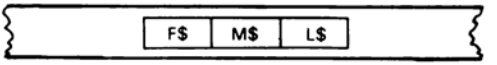
Come vedete ogni stringa è separata da un ritorno del carrello.

Vediamo ora come usare invece le virgole per separare le variabili di stringa.

Chiariamo subito che le virgole devono essere inserite esplicitamente perchè non sono ottenibili dai parametri dell'istruzione PRINT. Per esempio quando l'istruzione:

```
10 PRINT#1,F$,M$,L$
```

viene eseguita, il contenuto delle variabili F$, M$ e L$ sarà concatenato in un'unica stringa che sarà poi scritto sul nastro come segue:



Le virgole possono essere inserite tra i campi usando uno di questi due metodi:

1. Inserire il separatore tra virgolette:

```
PRINT#1,F$;",";M$;",";L$
```

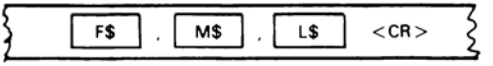
2. Usare la funzione CHR$ ( ):

```
PRINT#1,F$;CHR$(44);M$;CHR$(44);L$
```

separatore separatore

Dove ovviamente il valore della funzione CHR$(44) è il carattere virgola.

L'illustrazione delle stringhe separate da virgole è la seguente:



Nel programma successivo, chiamato NOMI.PRINT #, grazie ai separatori si evita che le tre stringhe F$, M$ e L$ siano conglobate in una sola:

NOMI.PRINT #

```
10 PRINT"◆◆ CREAZIONE DI DATI NUMERICI SU CASSETTA ◆◆":PRINT
20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ◆◆":PRINT
30 GET A$: IF A$="" THEN 30
40 PRINT"◆◆ APERTURA DEL FILE ◆◆":OPEN 1,1,2,"NUMERI"
50 FOR N=1 TO 10
60 PRINT N
70 PRINT#1,N
80 NEXT N
90 PRINT"◆◆ CHIUSURA DEL FILE ◆◆": CLOSE 1
100 END
```

Una facile regola, per ricordare come scrivere su una cassetta magnetica, è quella per cui i caratteri sono scritti sul nastro così come un'istruzione PRINT li scriverebbe sul display. Un ritorno del carrello viene registrato quando ci sarebbe un ritorno a

capo sullo schermo. Per avere una virgola di separazione dovete indicarla nell'istruzione PRINT # così come la indichereste in una PRINT per farla apparire sullo schermo tra due stringhe.

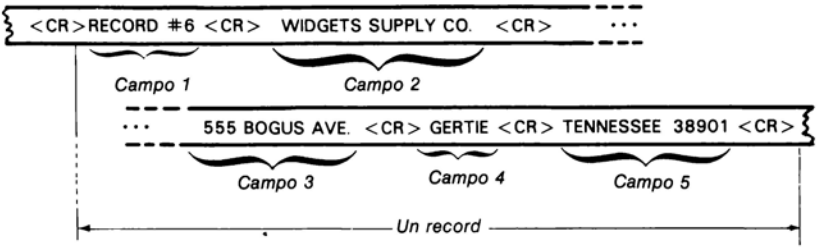
Nel prossimo esempio mostriamo come scrivere un elenco di indirizzi su una cassetta. Il nuovo programma ha nome MAIL.PRINT # e scrive un elenco di nome MAIL su cassetta. L'elenco MAIL sarà letto da un altro programma di nome MAIL.INPUT #.

In questo programma dimostrativo vogliamo porre in rilievo quali sono i passi necessari per scrivere dei record su una cassetta senza soffermarci troppo sugli aspetti di buona programmazione per l'ingresso dati. Il programma che desideriamo descrivere ha una logica di ingresso dei dati molto semplice, ma proprio per questo permette di concentrarci maggiormente sulla gestione della cassetta.

Ogni nome e indirizzo sono registrati come un unico record composto di cinque campi: 1) numero del record, 2) nome, 3) indirizzo stradale, 4) città e 5) stato e codice CAP. Ed ecco un esempio:

◆◆ RECORD #6 ◆◆	Campo 1	Un record
WIDGETS SUPPLY CO.	Campo 2	
555 BOGUS AVE.	Campo 3	
GERTIE	Campo 4	
TENNESSEE 38901	Campo 5	

Ovviamente questo non è il modo come i dati appaiono sulla cassetta che possiamo invece illustrare con il seguente disegno:



Il programma MAIL.PRINT # è scritto qui di seguito. Caricatelo nel vostro calcolatore e salvatelo su una cassetta. Prima di procedere stampatene un listing che apparirà con i caratteri standard della tastiera:

```
10 PRINT"□♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
20 PRINT"♠                        ♠"
30 PRINT"♠ INGRESSO INDIRIZZI ♠"
40 PRINT"♠                        ♠"
50 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
60 PRINT"■■■♠ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA ♠♠"
70 GET A$: IF A$="" THEN GO TO 70
80 PRINT"■■♠ APERTURA DEL FILE MAIL ♠♠":OPEN 1,1,2,"MAIL"
85 I=0
90 I=I+1
100 PRINT"□   ♠♠  NUM. PROGRESS. INDIRIZZO";I;"  ♠♠"
110 PRINT"                                    ■■"
120 PRINT" (PER FINIRE INGR. INDIR. BATTERE";CHR$(34);"FINE";CHR$(34);")"
130 PRINT"■■■": INPUT"1) NOME         ";NM$
140 IF NM$="FINE" THEN CLOSE 1: PRINT "□";"♠♠  FINE DEL PROGRAMMA  ♠♠": END
150 INPUT"2) LINEA IND. 1";A1$
160 INPUT"3) LINEA IND. 2";A2$
170 INPUT"4) LINEA IND. 3";A3$
180 INPUT"■■■■■■■  BATTERE N. CAMPO DA CAMBIARE (0 PER CONFER.)";X
190 IF X=0 THEN 220
200 IF X>=1 AND X<=4 THEN GOSUB 280
210 GOTO 180
220 PRINT#1,I
230 PRINT#1,NM$
240 PRINT#1,A1$
250 PRINT#1,A2$
260 PRINT#1,A3$
270 GOTO 90
280 PRINT"■": ON X GOTO 290,300,310,320
290 INPUT "1) NOME         ";NM$:RETURN
300 PRINT: INPUT "2) LINEA IND. 1";A1$: RETURN
310 PRINT"■": INPUT "3) LINEA IND. 2";A2$:RETURN
320 PRINT"■■": INPUT "4) LINEA IND. 3";A3$:RETURN
```

Le prime cinque linee (dalla 10 alla 50) descrivono brevemente la funzione del programma. Nella posizione successiva viene avvisato l'utente di montare la cassetta per i dati (linee 60 e 70).

In linea 80 viene aperto il file dati:

```
80 PRINT"■■♠ APERTURA DEL FILE MAIL ♠♠":OPEN 1,1,2,"MAIL"
```

L'archivio MAIL è aperto come file logico # 1 sulla prima unità a cassette ed è previsto che alla chiusura venga registrato un EOT. Il messaggio "APERTURA DEL FILE MAIL" è visualizzato sullo schermo prima di dare l'effettivo comando OPEN. Questo perchè l'apertura richiede alcuni secondi.

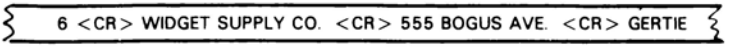
Ora il nastro è pronto per accettare i dati, ma prima di farlo è preferibile visualizzare ogni dato sullo schermo per controllare eventuali errori.

Le istruzioni dalla linea 130 alla 170 effettuano l'ingresso dei dati dalla tastiera e li visualizzano sullo schermo.

La variabile I alla linea 90 costituisce il contatore incrementale dei record ed è visualizzata alla linea 100. Le istruzioni dalla linea 130 alla 170 accettano le variabili NM$ (nome) e A1$, A2$ e A3$ (indirizzo) come campi singoli che sono separati dal carattere di ritorno carrello. Dopo l'ingresso dei dati nei quattro campi l'istruzione alla linea 180 chiede all'operatore se memorizzare il record o cambiare un campo in caso di errore. Se uno dei campi è errato l'operatore batte il numero corrispondente

(1-4) e il programma salta alla subroutine di correzione alla linea 280.

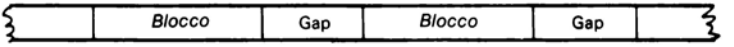
Mediante la variabile X indichiamo quale campo deve essere corretto e subito dopo ne diamo il nuovo valore. Possiamo correggere anche tutti e quattro i campi e alla fine dare il valore 0 alla X così che il programma prosegua dalla linea 220 fino alla 270. In queste ultime righe il record viene scritto nel file sulla cassetta come rappresenta la seguente illustrazione:



Assicuratevi che il numero di file logico indicato nell'istruzione PRINT # sia lo stesso del comando OPEN.

Dopo che il record è stato registrato, il programma ritorna alla linea 90 per preparare l'ingresso di un altro record. Alla fine di tutto l'archivio, l'operatore batte "FINE" alla richiesta del nome NM$. L'istruzione alla linea 140 chiude il file e scrive un EOT (come richiesto nella OPEN) quando appunto NM$ = "FINE".

Vi ricordiamo che il nastro non si muove ogni volta che registriamo un record, ma solo quando il buffer è pieno. Infatti, come vi abbiamo già detto, tutti i dati vengono prima inseriti in un buffer. Solo quando il buffer è pieno viene trasferito sulla cassetta come un blocco intero. Un blocco può contenere un solo record come anche una frazione o più record. Tra un blocco e l'altro il calcolatore CBM inserisce uno spazio chiamato "gap" come potete vedere in questo disegno:

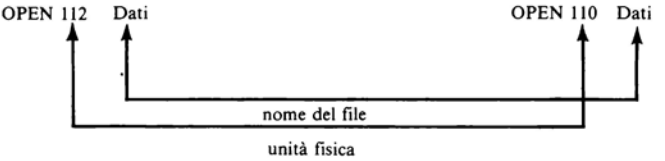


Lettura di numeri da cassetta magnetica

I punti da seguire nel programma sono tre:

1. Aprire il file di dati
2. Leggere il file
3. Chiudere il file.

Il file deve essere aperto in lettura con lo stesso nome che aveva ricevuto in scrittura. Il numero di file logico può essere diverso; l'indirizzo secondario deve essere 0:



Per leggere dalla cassetta possiamo usare due istruzioni: INPUT # e GET #. L'istruzione INPUT # permette la lettura di un record completo, con tutti i suoi

caratteri, fino al primo ritorno del carrello. Questo carattere di ritorno del carrello non viene però associato alle variabili in lettura. L'istruzione GET # permette invece di leggere un solo carattere alla volta compreso anche quello di ritorno del carrello.

Ricordatevi di chiudere il file dopo la lettura e di indicare lo stesso numero di canale.

```
OPEN  1,1,0, "DATI"
        ↕
CLOSE 1
```

Un buon metodo per chiudere un file in lettura è quello di cercare se esiste un segno di fine file EOF mediante l'esame della parola di stato (ST). Quando si scrive un file di dati alla sua fine viene posto un segno di EOF. Se in lettura attendiamo di incontrare l'EOF saremo sicuri di essere alla fine del file e così potremo chiuderlo. Il valore della parola di stato nel caso di EOF è 64. L'istruzione che ci permette di chiudere il file è allora:

```
IF ST=64 THEN CLOSE 1
```

Quando ST è pari a 64 il file viene chiuso.

Precedentemente abbiamo scritto il programma NUM.PRINT # per scrivere i numeri dall'1 al 10 in un file su cassetta chiamato NUMERI. Ora scriviamo un programma, a cui diamo nome NUM.INPUT #, per leggere i dieci numeri dal file NUMERI e visualizzarli sullo schermo.

Usiamo l'istruzione INPUT # che legge un campo alla volta sia numerico che di stringa.

Le prime istruzioni del programma NUM.INPUT # avvisano l'utente di caricare la cassetta e sono equivalenti alle prime istruzioni di NUM.PRINT #. Alla linea 30 c'è un ciclo di attesa per preparare la cassetta e premere RETURN:

```
10 PRINT"◆◆  LETTURA DI UN FILE NUMERICO  ◆◆": PRINT
20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ◆◆":PRINT
30 GET A$: IF A$="" THEN 30
```

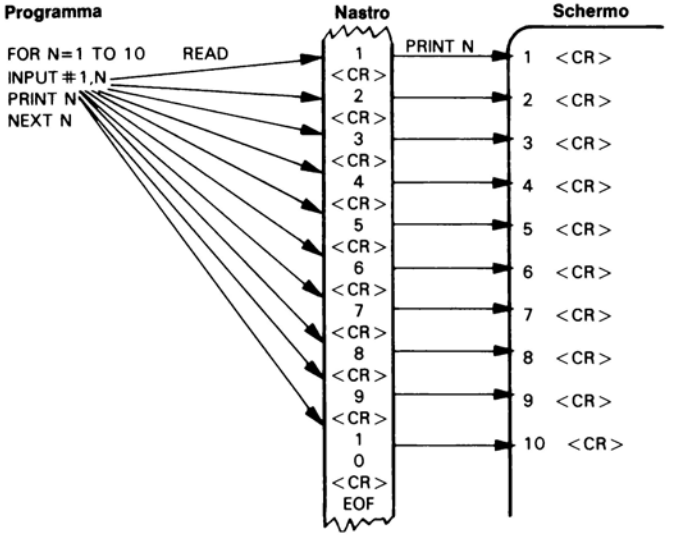
Prima della lettura il file deve essere aperto, come si è già altre volte detto. Alla linea 40 apriamo il file logico # 1, sul dispositivo fisico # 1, con l'indirizzo secondario 0 (apertura per sola lettura) e diamo il nome NUMERI al file.

```
40 PRINT"◆◆  APERTURA DEL FILE  ◆◆": OPEN 1,1,0,"NUMERI": PRINT
```

Un ciclo FOR NEXT legge i dieci numeri dal nastro e li visualizza sul display:

```
50 FOR I=1 TO 10
60 INPUT#1,N  ←———— Legge N dal nastro
70 PRINT N    ←———— Visualizza N sullo schermo
80 NEXT I
```

L'istruzione INPUT # 1 alla riga 60 legge un numero alla volta. L'esecuzione dell'intero ciclo può essere rappresentata dal seguente disegno:



Al termine della lettura il file deve essere chiuso.

```
90 PRINT"◆◆  CHIUSURA DEL FILE  ◆◆": CLOSE 1
100 END
```

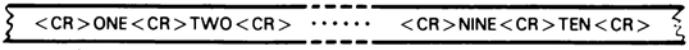
Vi diamo ora un listato completo del programma NUM.INPUT # seguito da un esempio di esecuzione.

NUM.INPUT #

```
 10 PRINT"◆◆  LETTURA DI UN FILE NUMERICO  ◆◆": PRINT
 20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ◆◆":PRINT
 30 GET A$: IF A$="" THEN 30
 40 PRINT"◆◆  APERTURA DEL FILE  ◆◆": OPEN 1,1,0,"NUMERI": PRINT
 50 FOR I=1 TO 10
 60 INPUT#1,N
 70 PRINT N
 80 NEXT I
 90 PRINT"◆◆  CHIUSURA DEL FILE  ◆◆": CLOSE 1
 100 END
READY.
```

```
♦♦  LETTURA DI UN FILE NUMERICO  ♦♦

♦♦  INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ♦♦

♦♦  APERTURA DEL FILE  ♦♦

PRESS PLAY ON TAPE # 1
OK

 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
♦♦  CHIUSURA DEL FILE  ♦♦
```

L'istruzione INPUT # permette di leggere anche campi che contengano variabili di stringa. Il programma PAROL.PRINT # scriveva dieci stringhe sulla cassetta nel file PAROLNUM. Il file PAROLNUM può essere così disegnato:



Per leggere i campi del file PAROLNUM usiamo INPUT # seguita da una variabile di stringa anzichè numerica. Con delle piccole modifiche è possibile quindi cambiare il programma, per la LETTURA DI UN FILE NUMERICO, nel programma per la lettura di numeri, espressi in lettere, contenuti nel file PAROLNUM. Le modifiche si devono fare alla linea 40 (nome del file) e alla linea 60 (variabile di INPUT). Il nuovo programma è il seguente:

```
10 PRINT"♦♦  LETTURA DI UN FILE DI STRINGHE NUMERICHE  ♦♦": PRINT
20 PRINT"♦♦ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ♦♦":PRINT
30 GET A$: IF A$="" THEN 30
40 PRINT"♦♦  APERTURA DEL FILE  ♦♦": OPEN 1,1,0,"PAROLNUM": PRINT
50 FOR I=1 TO 10
60 INPUT#1,N$
70 PRINT N$
80 NEXT I
90 PRINT"♦♦  CHIUSURA DEL FILE  ♦♦": CLOSE 1
100 END
```

```
◆◆  LETTURA DI UN FILE DI STRINGHE NUMERICHE  ◆◆

◆◆  INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ◆◆

◆◆  APERTURA DEL FILE  ◆◆

PRESS PLAY ON TAPE # 1
OK

UNO
DUE
TRE
QUATTRO
CINQUE
SEI
SETTE
OTTO
NOVE
DIECI
◆◆  CHIUSURA DEL FILE  ◆◆
```

Passiamo ora ad un altro programma cha abbiamo già incontrato: NOMI-.PRINT # che scrive dei nomi in un file chiamato NOMI. Ricordiamo che i singoli nomi sono scritti ognuno in tre campi: F$, M$ e L$ separati tra loro da virgole. Nel disegno seguente diamo un'idea figurativa dei dati sul nastro.

HEADLY, GEORGE, JOYCE<CR>CAROL, A., SMITH<CR>

Se non ci fossero le virgole di separazione, i campi sarebbero letti come una singola stringa e sullo schermo apparirebbero così:

```
HEADLYGEORGEJOYCE
CAROLA.SMITH
```

Qui di seguito diamo il listato di un programma per leggere i nomi del file NOMI. L'istruzione INPUT # alla linea 60 legge tutti i campi sino al carattere di ritorno carrello successivo. I campi compresi tra due ritorni carrello sono separati da virgole. Giacchè in questo caso sono tre, nella istruzione INPUT # devono essere presenti tre nomi di variabili. L'istruzione PRINT alla linea 70 visualizza le tre stringhe su una singola riga separandole con uno spazio.

```
10 PRINT"◆◆  LETTURA DEL FILE NOMI  ◆◆": PRINT
20 PRINT"◆◆ INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ◆◆":PRINT
30 GET A$: IF A$="" THEN 30
40 PRINT"◆◆  APERTURA DEL FILE  ◆◆": OPEN 1,1,0,"NOMI": PRINT
50 FOR I=1 TO 4
60 INPUT#1,F$,M$,L$
70 PRINT F$;" ";M$;" ";L$
80 NEXT J
90 PRINT"◆◆  CHIUSURA DEL FILE  ◆◆": CLOSE 1
100 END
```

```
◆◆  LETTURA DEL FILE NOMI  ◆◆

◆◆  INSERIRE LA CASSETTA E PREMERE <RETURN> QUANDO E' PRONTA  ◆◆

◆◆  APERTURA DEL FILE  ◆◆

PRESS PLAY ON TAPE # 1
OK

PASQUALE ROSSI
GIGI RIVA
GUIDO ROSSI
ANDREA BIANCHI

◆◆  CHIUSURA DEL FILE  ◆◆
```

Passiamo ora al programma dimostrativo per leggere l'elenco di indirizzi che era stato scritto nel file MAIL dal programma MAIL.PRINT #. Ogni record contiene cinque campi: numero del record, nome, strada, città e stato con CAP. Qui di seguito diamo un esempio di record del file MAIL:

◆◆ RECORD #6 ◆◆	Campo 1	Un record
WIDGETS SUPPLY CO.	Campo 2	
555 BOGUS AVE.	Campo 3	
GERTIE	Campo 4	
TENNESSEE 38901	Campo 5	

Riportiamo il programma MAIL.INPUT #. Caricatelo da tastiera nel vostro calcolatore e salvatelo su una cassetta. Provate poi a listarlo per potere seguire la nostra spiegazione.

MAIL.INPUT #

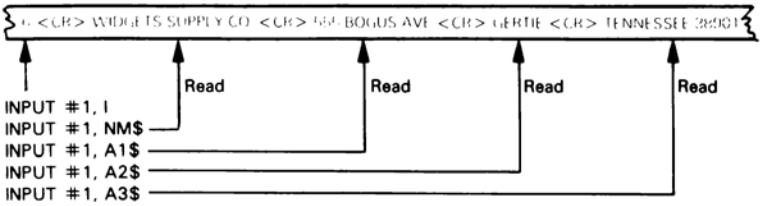
```
10 PRINT"◆◆◆
```

e 70 vengono date le istruzioni all'operatore per caricare la cassetta magnetica. Alla linea 80 il file MAIL viene aperto con file logico # 1 sulla unità a cassette # 1. L'indirizzo secondario deve essere 0 per indicare l'operazione di lettura.

```
30 PRINT"♠♠  APERTURA DEL FILEMAIL  ♠♠":OPEN 1,1,0,"MAIL"
```

L'istruzione alla linea 100 esamina la parola di stato (ST) per individuare il segno di fine del file EOF. Se ST = 64 (che significa appunto che si è incontrata la fine del file), il file viene chiuso alla linea 9999. Facciamo presente che la parola ST deve essere controllata prima della lettura per evitare che si tentino di leggere dati ove non sono stati registrati.

Le istruzioni dalla linea 110 alla 150 leggono i dati mediante la INPUT #. Siccome ogni campo, nella fase di scrittura, era stato separato da un carattere di ritorno del carello, adesso per la lettura bisogna usare una INPUT # diversa per ogni campo. I nomi delle stringhe usati in scrittura possono essere diversi dai nomi usati in lettura. Per esempio possiamo aver scritto con la variabile X$ e leggere invece con A$. Non fa alcuna differenza infatti perchè i nomi delle variabili non sono ne memorizzati ne trasferiti da un programma all'altro.



I dati in lettura sono depositati nel buffer di memoria. Essi non sono visualizzati sino a che non lo si ordini tramite il programma.

Questa visualizzazione è fatta alle linee da 160 a 200. La linea 210 muove il cursore di quattro righe in basso.

```
160 PRINT"♠♠ RECORD #";I$;" ♠♠"
170 PRINT"NAME ";TAB(9);NM$
180 PRINT"ADDR ";TAB(9);A1$
190 PRINTTAB(9);A2$
200 PRINTTAB(9);A3$
210 PRINT"
```

L'uscita sullo schermo apparirà come segue:

```
♦♦ RECORD #6 ♦♦

NOME:     WIDGETS SUPPLY CO.
INDIR:    555 BOGUS AVE.
          GERTIE
          TENNESSEE  38901
```

Dopo che i quattro campi sono stati visualizzati, il programma chiede all'operatore se desidera leggere il record successivo:

```
220 INPUT"BATTERE 'Y' PER LEGGERE UN ALTRO RECORD";A$: IF A$="Y" GOTO 100
```

Se è richiesta una successiva lettura il programma salta alla riga 100 e prosegue sino a che incontra un EOF. Nel caso invece che si desideri terminare la lettura o si incontri un EOF il file viene chiuso e il programma termina.

Nella Figura 6-4 trovate il diagramma a blocchi del programma MAIL.INPUT # e quindi un esempio di esecuzione:

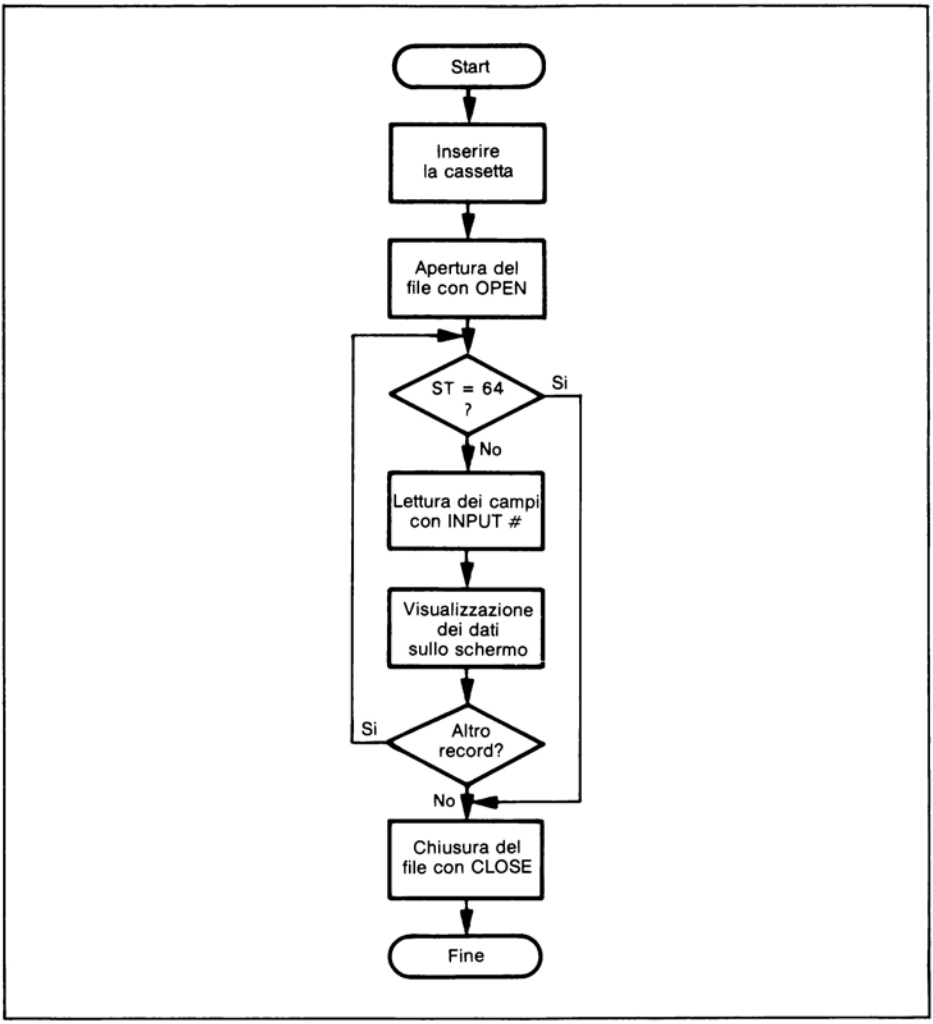


Figura 6-4: MAIL. INPUT #

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                            ♠
♠  LETTURA DEL FILE MAIL     ♠
♠                            ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠

♠♠  PREMERE <RETURN> QUANDO LA CASSETTA E' CARICATA  ♠♠

♠♠  APERTURA DEL FILE MAIL  ♠♠

PRESS PLAY ON TAPE #1
OK


♠♠  LETTURA DEL FILE MAIL  ♠♠


♠♠ RECORD # 1 ♠♠

NOME:           ACME MANUFACTURING CO.
INDIR:          1235 MAIN ST.
                DOWNTOWN
                IL  62501

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD

♠♠ RECORD # 2 ♠♠

NOME:           BENJAMIN FRANKLIN
INDIR:          12 LIBERTY TOWER
                PHILADELPHIA
                PA  16524

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD

♠♠ RECORD # 3 ♠♠

NOME:           NEIL ARMSTRONG
INDIR:          597 SEA OF TRANQUILITY AVE.
                EARTHVIEW
                LUNAR  000000

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD

♠♠ RECORD # 4 ♠♠

NOME:           MAMMOTH DISTRIBUTION CO.
INDIR:          INDUSTRIAL PARK
                CITY OF INDUSTRY
                CA  92425

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD
```

```
♠♠ RECORD # 5 ♠♠

NOME:           HENRY MUSCATEL
INDIR:          819 OAK ST.
                NAPA
                CA  95303

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD

♠♠ RECORD # 6 ♠♠

NOME:           WIDGET SUPPLY CO.
INDIR:          555 BOGUS
                GERTIE
                TENNESSEE  38901

BATTERE 'Y' PER LEGGERE UN ALTRO RECORD
♠♠  FINE DEL FILE MAIL; PROGRAMMA TERMINATO  ♠♠
```

Durante la esecuzione di MAIL.INPUT # non spaventatevi se vedete il calcolatore fermarsi per alcuni secondi. Guardate la cassetta e la vedrete in movimento. Infatti il calcolatore deve periodicamente riempire il suo buffer di memoria di 191 byte con i dati letti dalla cassetta prima di continuare il programma.

Prima di procedere oltre desideriamo farvi notare che la linea 220 non costituisce un buon modo di programmare. Infatti essa prevede che venga visualizzato un altro record solo se l'operatore batte una Y. Ma se viene battuto un qualunque altro tasto, anche solo per errore, il programma si ferma. Un buon programma prevede che si batta un carattere di approvazione e un carattere diverso per la negazione. Per esempio Y (yes) e N (no); mentre ogni altro carattere è ignorato ed in tal caso la richiesta se procedere viene ripetuta. Provate a riscrivere la linea 220 secondo questi criteri.

Un altro metodo per leggere i file di dati impiega l'istruzione GET #:

	GET # f, var	
dove:	f	è il numero di file logico (corrisponde al numero di file della OPEN e CLOSE)
	var	è il nome della variabile da leggere.

GET # legge un carattere alla volta dal file; essa è simile alla istruzione GET che accetta un carattere alla volta dalla tastiera.

GET # legge sia caratteri che segnali di delimitazione del file come anche ogni altra cosa scritta sul nastro. Essa è particolarmente utile quando si desidera leggere qualunque dato sia stato registrato su un nastro difettoso per poter capire che cosa non funziona. GET # permette di confrontare individualmente ogni carattere con dei valori specifici e cioè di effettuare una vera e propria identificazione di caratteri.

Due programmi dimostrativi illustreranno come leggere e stampare un intero file, compresi i delimitatori del file, e come stampare il file di dati MAIL ripartito in ogni singolo record.

Il programma MAIL.GET # 1, che qui sotto riportiamo, legge un carattere alla volta del file MAIL e ne visualizza il contenuto sullo schermo:

MAIL.GET # 1

```
10 PRINT"♠♠♠
```

Ecco un esempio di esecuzione:

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                          ♠
♠  LETTURA DEL FILE MAIL   ♠
♠    VERSIONE  GET #       ♠
♠                          ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠

♠♠  PREMERE <RETURN> QUANDO LA CASSETTA E' CARICATA  ♠♠

♠♠  APERTURA DEL FILE MAIL  ♠♠

PRESS PLAY ON TAPE #1
OK

♠♠  FILE MAIL  ♠♠





♠♠  FINE DEL FILE MAIL; PROGRAMMA TERMINATO  ♠♠

 1 ※ACME MANUFACTURING CO.※1235 MAIN ST.
※DOWNTOWN※IL    62501※ 2 ※BENJAMIN FRANKL
IN※12 LIBERTY TOWER※PHILADELPHIA    16524
※ 3 ※NEIL ARMSTRONG※597 SEA OF TRANQUILI
TY※EARTHVIEW※LUNAR    00000※ 4 ※MAMMOTH D
ISTRIBUTION CO.※INDUSTRIAL PARK※CITY OF
INDUSTRY※CA    92425※ 5 ※HENRY MUSCATEL※3
19 OAK ST.※NAPA※CA    95303※ 6 ※WIDGET SU
PPLY CO.※555 BOGUS AVE.※GERTIE※TENNESSEE
    38901※
```

Il secondo programma dimostrativo, MAIL.GET # 2, legge il file MAIL e lo visualizza diviso in record. Il listato di MAIL.GET # 2 è il seguente:

MAIL.GET # 2

```
10 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
20 PRINT"♠                          ♠"
30 PRINT"♠  LETTURA DEL FILE MAIL   ♠"
35 PRINT"♠    VERSIONE  GET #       ♠"
40 PRINT"♠                          ♠"
50 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠":PRINT:PRINT
60 PRINT"♠♠  PREMERE <RETURN> QUANDO LA CASSETTA E' CARICATA  ♠♠":PRINT
70 GET A$: IF A$="" THEN 70
80 PRINT"♠♠  APERTURA DEL FILE MAIL  ♠♠": PRINT: OPEN1,1,0,"MAIL"
90 PRINT:PRINT"♠♠  FILE MAIL  ♠♠":PRINT
95 F=0:R=0
100 IF ST=64 THEN 9999
110 GET#1,X$
120 IF X$=CHR$(13) THEN F=F+1
130 PRINT X$;
140 IF F>=5 THEN GOSUB 160
150 GOTO 100
160 PRINT
170 R=R+1
180 IF R>2 THEN PRINT"PREMERE 'Y' PER NUOVO GRUPPO DI RECORD";:INPUT A$
185 IF A$="Y" THEN R=0
190 F=0: PRINT: RETURN
9999 PRINT"♠♠♠♠  FINE DEL FILE MAIL; PROGRAMMA TERMINATO  ♠♠":CLOSE1:END
```

Introducete, tramite tastiera, MAIL.GET # 2, poi salvatelo (SAVE) e verificatelo (VERIFY) su una cassetta. Alla fine provate anche a listarlo.

Le linee da 10 a 100 di MAIL.GET # 2 sono eguali a quelle di MAIL.GET # 1 e in ambedue i programmi ne descrivono i compiti e aprono il file MAIL in lettura.

La differenza tra i due programmi è alla linea 120 dove, invece di assegnare X$ il carattere FULL GRID (se X$ = CHR$ (13)), viene invece incrementato un contatore F. Quando il programma MAIL.PRINT # aveva scritto il file, un carattere di ritorno del carrello aveva segnato la fine di ogni campo. Siccome ogni record è composto da cinque campi, il programma MAIL.GET # 2 conta cinque campi alla volta per determinare ogni record. Alla linea 140 viene appunto fatto un salto ad una subroutine ogni qualvolta F è uguale a 5.

L'istruzione alla linea 160 inserisce una linea vuota tra due record successivi. Alla linea 170 viene incrementata la variabile R quale contatore di record. Alla linea 180 viene controllato se sono stati letti più di 2 record. Infatti lo schermo del display non può contenere più di tre record per cui viene chiesto all'operatore se desidera che altri tre record vengano visualizzati. Se la risposta è positiva i due contatori, R per i record e F per i campi, vengono azzerati prima che il programma prosegua alla linea 100. Il programma continua quindi a stampare tre record alla volta sino a che l'operatore non dia un carattere diverso da Y o venga incontrato lo stato ST = 64; in tal caso il file viene chiuso e il programma terminato. In Figura 6-5 trovate lo schema a blocchi di questo programma.

Sebbene GET # sia simile a INPUT #, è più difficile stabilire il formato delle stampe in uscita con GET # se sono desiderate spaziature o intitolazioni. Proprio come abbiamo in precedenza testato X$ con CHR$(13) così bisognerebbe fare con ogni delimitatore di campo o carattere per ottenere un determinato formato di stampa.

Qui di seguito vi diamo un esempio di esecuzione di MAIL.GET # 2.

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                           ♠
♠  LETTURA DEL FILE MAIL    ♠
♠    VERSIONE  GET #        ♠
♠                           ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠

♠♠  PREMERE <RETURN> QUANDO LA CASSETTA E' CARICATA  ♠♠

♠♠  APERTURA DEL FILE MAIL  ♠♠

PRESS PLAY ON TAPE #1
OK

♠♠  FILE MAIL  ♠♠

 1
ACME MANUFACTURING CO.
1235 MAIN ST.
DOWNTOWN
IL    62501
```

```
 2
BENJAMIN FRANKLIN
12 LIBERTY TOWER
PHILADELPHIA
PA    16524


 3
NEIL ARMSTRONG
597 SEA OF TRANQUILITY
EARTHVIEW
LUNAR    00000

PREMERE 'Y' PER NUOVO GRUPPO DI RECORD?Y
 4
MAMMOTH DISTRIBUTION CO.
INDUSTRIAL PARK
CITY OF INDUSTRY
CA    92425


 5
HENRY MUSCATEL
819 OAK ST.
NAPA
CA    95303


 6
WIDGET SUPPLY CO.
555 BOGUS AVE.
GERTIE
TENNESSEE    38901

PREMERE 'Y' PER NUOVO GRUPPO DI RECORD?Y

♠♠  FINE DEL FILE MAIL; PROGRAMMA TERMINATO  ♠♠
```

FORMATO DEI FILE SU CASSETTA

La descrizione dei file di dati, presentata all'inizio di questo capitolo, è una introduzione accurata e concettualmente valida di come i calcolatori gestiscono in generale gli archivi di dati. **La ripartizione dei file (o archivi) in record e campi è un metodo generale che si può anche mantenere con i calcolatori CBM ricorrendo ad una appropriata programmazione.** Fin che vi è possibile cercate pure di mantenere questa strutturazione dei dati, ma sappiate che i file su cassette CBM adottano altre strutture che appunto desideriamo illustrarvi.

Ogni campo numerico deve essere seguito da un carattere di ritorno del carrello (CHR$(13)). Di conseguenza un file costituito da soli numeri può essere visto come

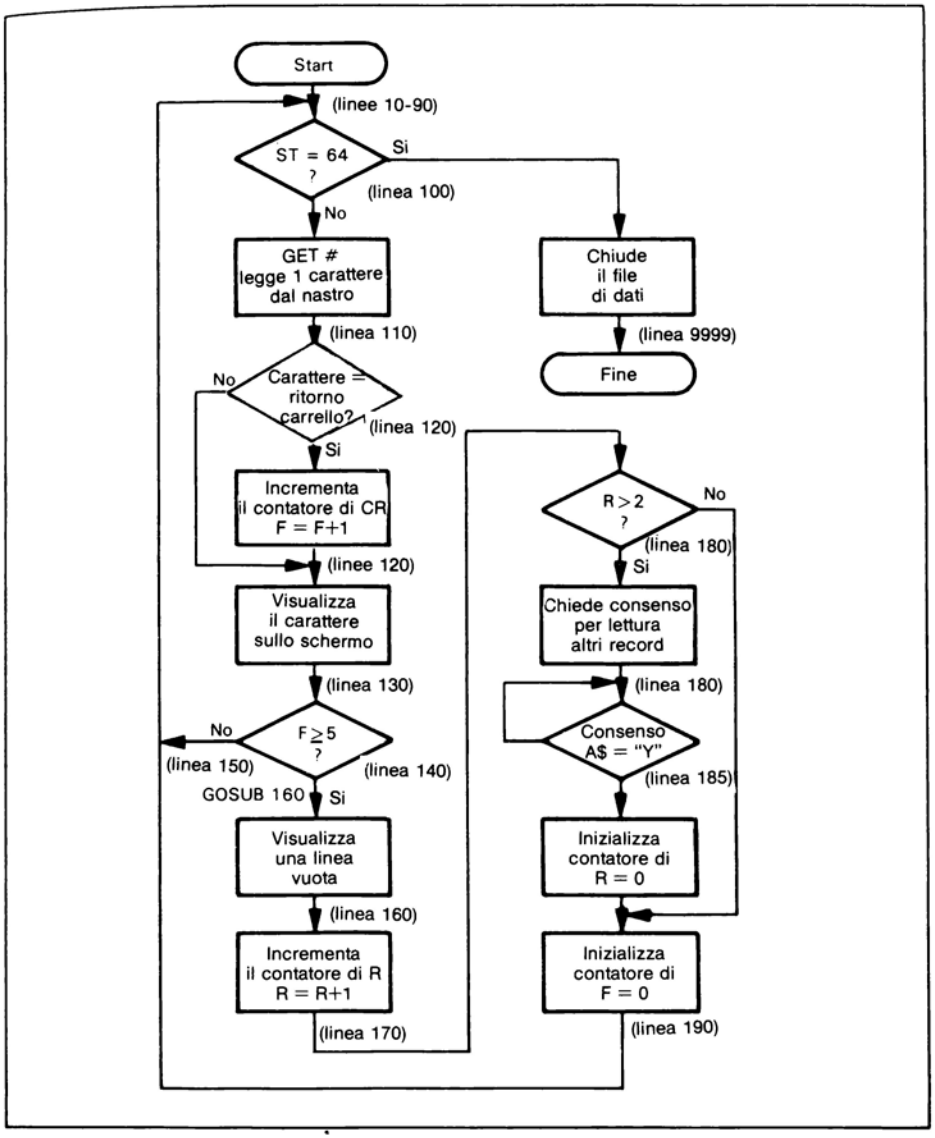


Figura 6-5: Formato di visualizzazione usando GET #.

una sequenza di numeri separati da caratteri di ritorno del carrello:

N<CR>N<CR>N<CR>N<CR>N<CR>

Non c'è niente quindi che permetta di riconoscere record o campi. Dipende solo dalle necessità del programmatore di definire delle sequenze ripetitive da interpretarsi come record.

Le variabili di stringa possono invece essere divise in campi e record. I campi possono essere separati da virgole (CHR$ (44)) mentre i record saranno separati da caratteri di ritorno del carrello (CHR$(13)). Ecco un esempio di file di sole stringhe con cinque campi per record:

```
<CR>S<,>S<,>S<,>S<,>S<CR>S<,>S<,>S<,>S<CR>
```

Attenzione però che se usate la virgola per separare i campi e il ritorno carrello per separare i record, allora dovrete usare una unica istruzione INPUT # per leggere tutto il record.

L'uso di virgole per separare i campi non è obbligatorio; potete usare anche in questo caso il ritorno carrello. **Noi vi consigliamo di usare sempre il ritorno carrello per separare qualunque dato e organizzare i dati in campi e record a livello di programma.**

La logica di programmazione per strutturare i file in campi e record è spesso molto semplice ed evidente di per sè. Prendiamo come esempio l'elenco degli indirizzi "mailing list" e vediamo che ogni record è costituito dal nome e dall'indirizzo mentre la ripartizione in nome, cognome, via, città, ecc. costituisce quella in campi. La ripartizione in campi può esser fatta in molti modi e dipende solo dalle necessità del programmatore di sceglierne una piuttosto che un'altra. Le uniche difficoltà possono essere dovute alla sintassi delle istruzioni PRINT # e INPUT #.

Per mostrarvi queste eventuali difficoltà di sintassi vi proponiamo delle variazioni di un programma molto semplice:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J
100 PRINT J
110 NEXT
120 CLOSE 1
132 STOP
```

L'istruzione OPEN alla linea 10 apre il file logico 1 e seleziona il drive 1 per operazioni di scrittura. Il ciclo FOR-NEXT alle linee 20, 30 e 40 scrive 10 numeri sulla cassetta ognuno seguito dal carattere di ritorno carrello, come voluto dalla PRINT # della linea 30 e in analogia ad una PRINT che porterrebbe il carrello a

capo. Il file logico è chiuso alla linea 50. I dieci numeri potrebbero essere visti sul nastro come appare sul disegno seguente:

Le istruzioni dalla linea 70 alla 120 leggono e visualizzano i dieci numeri precedentemente registrati.

Proviamo a eseguire questo programma e vediamo che cosa succede.

Prendete una cassetta e caricatela nel drive 1. Assicuratevi anche che nessun tasto del registratore sia premuto.

Listate il programma per assicurarvi che sia caricato correttamente e poi date il comando RUN. Sullo schermo apparirà il seguente messaggio:

```
PRESS PLAY AND RECORD ON TAPE #1
```

Premete i due tasti sul drive 1; il calcolatore vi risponderà:

```
PRESS PLAY AND RECORD ON TAPE #1
OK
```

I dieci numeri sono ora registrati sulla cassetta. Al termine il drive si ferma e appare il seguente messaggio:

```
BREAK IN 60
READY
```

Il cursore riappare e lampeggia. L'istruzione STOP alla linea 60 è quella che causa l'interruzione. Premete allora il tasto STOP del registratore per sollevare gli altri due tasti PLAY e RECORD. Per riavvolgere il nastro premete REWIND e alla fine nuovamente STOP. Eseguite quindi la seconda metà del programma battendo il comando:

```
GOTO 70
```

Sullo schermo apparirà il messaggio PRESS PLAY ON TAPE # 1; premete quindi il tasto PLAY del primo drive. Il calcolatore vi risponde OK:

```
PRESS PLAY ON TAPE 1
OK
```

I dieci numeri da 101 a 110 vengono quindi letti e visualizzati verticalmente sullo schermo:

```
101
102
103
104
105
106
107
108
109
110
BREAK IN 130
READY
```

I dieci numeri sono visualizzati verticalmente perchè l'istruzione PRINT alla linea 100 esegue una visualizzazione alla volta.

Il messaggio finale è dovuto all'istruzione STOP della linea 130.

```
BREAK IN 130
READY
```

Se per errore vi siete dimenticati di riavvolgere il nastro prima di battere l'istruzione GOTO 70, allora il drive cercherà di leggere vecchi numeri sulla cassetta fino alla fine del nastro. Dovete quindi fermare la cassetta ed il programma; riavvolgere il nastro, ma prima di rimettere in esecuzione il programma è corretto chiudere il file logico 1 in modo immediato:

```
CLOSE 1
```

Quindi:

```
GOTO 70
```

Aggiungiamo ora un punto e virgola alla istruzione PRINT della linea 100:

```
100 PRINT J;
```

Riavvolgiamo il nastro e battiamo GOTO 70.

Sullo schermo apparirà nuovamente PRESS PLAY ON TAPE # 1. Appena premuto PLAY il calcolatore vi risponde OK. Dopo una breve pausa, in cui vengono letti i dieci numeri dalla cassetta, sul display riappariranno i numeri, ma questa volta tutti su una riga:

```
101  102  103  104  105  106  107  108  109  110
BREAK IN 130
READY
```

Proviamo ora a modificare le linee dalla 80 alla 110 così che i dieci numeri siano

letti dalla cassetta mediante una sola istruzione INPUT:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
50 CLOSE 1
60 STOP
70 OPEN 1
80 INPUT#1,N(1),N(2),N(3),N(4),N(5),N(6),N(7),N(8),N(9),N(10)
90 FOR I=1 TO 10
100 PRINT N(I);
110 NEXT
120 CLOSE 1
130 STOP
```

Ancora una volta riavvolgete il nastro ed eseguite la seconda parte del programma battendo GOTO 70.

Analogamente ai casi precedenti il calcolatore vi avviserà con PRESS PLAY ON TAPE # 1. Appena premete PLAY i dieci numeri verranno letti e poi visualizzati su una stessa riga come per il programma precedente. Possiamo quindi dedurre che non fa alcuna differenza se eseguiamo una sola INPUT #, con più variabili elencate nella sua lista parametrica, o ripetiamo più volte INPUT # con una sola variabile.

Proviamo ora altri segni di punteggiatura come caratteri separatori. Come primo esempio inseriamo, nella fase di registrazione sulla cassetta, dei punti e virgola come illustrato nel programma che segue:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
45 C$=CHR$(59)
46 PRINT#1,M(1);C$;M(2);C$;M(3);C$;M(4);C$;M(5)
47 PRINT#1,M(6);C$;M(7);C$;M(8);C$;M(9);C$;M(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J
100 PRINT J
110 NEXT
120 CLOSE 1
130 STOP
```

Dove CHR$(59) rappresenta il punto e virgola. Riavvolgete il nastro e portatelo all'inizio della zona magnetica. Inserite la cassetta nel drive 1 e controllate che tutti i tasti siano sollevati. Battete alla fine RUN e quando vi verrà richiesto premete PLAY e RECORD. I dati saranno memorizzati ed il seguente messaggio vi sarà visualizzato:

```
BREAK IN 60
READY
▒
```

Riavvolgete il nastro e battete GOTO 70. Come già sapete dovete poi premere

PLAY, ma questa volta i dati non appariranno sullo schermo e anzi vi sarà data una segnalazione di errore:

```
FILE DATA ERROR IN 90
READY
```

L'errore consiste nell'aver usato il punto e virgola per separare dati numerici: per i dati numerici potete usare solamente il ritorno carrello. Per separare invece le stringhe potete usare sia le virgole che il ritorno carrello. Provate infatti a fare questa modifica nell'ultimo programma:

```
5 DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 READ M$(I)
40 NEXT
45 C$=CHR$(44)
46 PRINT#1,M$(1);C$;M$(2);C$;M$(3);C$;M$(4);C$,M$(5)
47 PRINT#1,M$(6);C$;M$(7),C$;M$(8);C$;M$(9);C$;M$(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J$
100 PRINT J$
110 NEXT
120 CLOSE 1
130 STOP
```

Riavvolgete la cassetta e fate avanzare il nastro sino a che la superficie magnetica appaia sotto la finestra di lettura. Inserite poi la cassetta nel drive 1 e battete RUN. Quando vi verrà richiesto premete PLAY e RECORD del registratore 1. I dati saranno registrati e sullo schermo apparirà il messaggio:

```
BREAK IN 60
READY
```

riavvolgete quindi la cassetta e battete GOTO 70.

Premete PLAY; vedrete visualizzare le stringhe 1 e 6 seguite però da un messaggio di errore:

```
STRING TOO LONG ERROR IN 90
READY
```

Che casa è accaduto? L'errore è nell'istruzione INPUT # della linea 90. Una istruzione INPUT # legge infatti tutti i campi di stringa sino al primo carattere di ritorno carrello. Di conseguenza i valori da M$(1) a M$(5) sono stati fatti entrare alla prima esecuzione della linea 90, ma solo il primo valore M$(1) è stato assegnato a J$ poichè la virgola viene interpretata come un separatore di campi e non come la fine di un record. Alla seconda esecuzione della linea 90 entrano gli altri valori da M$(6) a M$(10) che sono visti come falsi campi tra due ritorni di carrello. Anche

questa volta, per lo stesso motivo, solo il valore M$(6) viene assegnato a J$. Quando la linea 90 viene eseguita per la terza volta non ci sono più dati da poter leggere per cui viene dato un errore di file come abbiamo visto più sopra.

Per risolvere il problema dobbiamo allora inserire l'istruzione INPUT # con lo stesso numero di variabili che erano listate nell'istruzione PRINT #. Se modifichiamo allora il nostro programma:

```
5 DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 READ M$(I)
40 NEXT
45 C$=CHR$(44)
46 PRINT#1,M$(1);C$;M$(2);C$;M$(3);C$;M$(4);C$;M$(5)
47 PRINT#1,M$(6);C$;M$(7);C$;M$(8);C$;M$(9);C$;M$(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 INPUT#1 N$(1),N$(2),N$(3),N$(4),N$(5)
90 INPUT#1 N$(6),N$(7),N$(8),N$(9),N$(10)
100 FOR I=1 TO 10
105 PRINT N$(I);" ";
110 NEXT
120 CLOSE 1
130 STOP
```

e ripetiamo correttamente le fasi di scrittura dei dati e poi di lettura sulla cassetta, otterremo sul display:

```
ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE TEN
BREAK IN 130
READY
```

Ci sono altri casi, di scrittura e lettura di dati su cassetta, che meritano di essere da voi stessi provati.

Può una singola INPUT # leggere più variabili di stringa separate da caratteri di ritorno carrello? Per fare questa prova cambiate la linea 45 dell'ultimo programma così che a C$ sia assegnato il valore CHR$(13).

È consentito mescolare dati numerici con stringhe in un singolo file? Per questa prova, oltre alle stringhe M$(I), aggiungete nell'ultimo programma alla nuova linea 35 il vettore numerico M(I):

```
35 M(I)=I+100
```

Provate allora le possibili combinazioni di PRINT # alle linee 46 e 47 così da ottenere poi la corretta lettura con le linee 80 e 90.

FILE SU DISCHETTO

Oltre che sulle cassette i file, sia di programmi che di dati, possono essere registrati anche su dischetti. I file di programmi memorizzano programmi in BASIC; i file di dati memorizzano valori numerici e stringhe.

Esistono tre tipi di file di dati su dischetti:

1. **File sequenziali**. Questi file memorizzano i dati in maniera molto compatta, ma hanno capacità di accesso molto limitate.
2. **File relativi**. Richiedono più spazio sui dischetti che i file sequenziali, ma permettono un più facile accesso e una gestione più efficiente.
3. **File ad accesso diretto**. La lora struttura viene definita nell'ambito del vostro stesso programma.

I file di programmi, i file sequenziali di dati e i file di dati relativi sono descritti in questo capitolo. I file di dati ad accesso diretto saranno invece descritti nel capitolo 7.

Confronto tra i file su cassetta e quelli su dischetto

La gestione dei file su dischetto differisce notevolmente da quella dei file su cassetta per due ragioni:

1. Il tempo medio di accesso ai dati su dischetto è molto più breve rispetto a quello relativo alle cassette.
2. Non esiste un "inizio" o "fine" della superficie del dischetto come invece avviene per un nastro magnetico. Si può accedere con eguale facilità a qualunque dato su un dischetto, mentre invece su una cassetta è necessario far avanzare o riavvolgere il nastro.

Le gestioni dei file su cassetta o su dischetto differiscono notevolmente per il diverso metodo di accesso e per il diverso formato di registrazione dei dati. Queste differenze non riguardano gli aspetti meccanici delle due unità; anzi la velocità di rotazione del disco è confrontabile a quella di trascinamento del nastro.

Le cassette registrano i dati su una traccia continua lunga tanto quanto il nastro stesso; le testine di lettura e di scrittura sono ferme mentre il nastro viene mosso sotto di esse per accedere a qualunque dato.

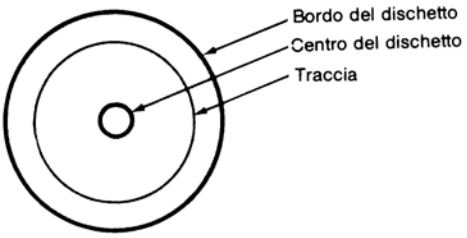
I dischetti sono invece completamente diversi perchè registrano i dati su tracce circolari concentriche. La testina di lettura e scrittura è posta su un braccio mobile e può spostarsi su una qualunque traccia. Il dischetto è tenuto sempre in rotazione per portare un settore qualunque delle tracce sotto la testina.

Per usare i dischetti non è necessario conoscere i dettagli della registrazione sulla superficie del dischetto stesso, ma una conoscenza in generale permette di sfruttare meglio le possibilità di questi supporti. Diamo quindi una breve descrizione di come i dati sono registrati sui dischetti.

REGISTRAZIONE DEI DATI SU DISCHETTO

Come abbiamo accennato i dati sono registrati su tracce concentriche ed ogni traccia è divisa in settori angolari.

Per immaginare una traccia provate a disegnare un cerchio che rappresenti un dischetto e poi al suo interno una circonferzenza concentrica. Questa circonferenza è appunto una traccia, come potete vedere anche in questo disegno:



Non tutte le unità a disco scrivono sullo stesso numero di tracce. È possibile poi scrivere su una sola superficie o su ambedue le superfici. **I drive CBM 2040 e 8050 scrivono su una sola superficie; il modello 2040 su 35 tracce e il modello 8050 su 77 tracce.** In Tabella 6-3 trovate le specifiche di queste unità.

Per rendere più semplice l'accesso ai dati su tracce diverse, il drive non scrive egualmente su tutti i punti di una traccia. Se i dati fossero registrati lungo tutta una traccia completa non vi sarebbero due tracce con la stessa quantità di informazione giacchè le tracce sono concentriche e non hanno la stessa lunghezza. **Per risolvere questo problema le tracce sono divise in settori. Ogni settore contiene la stessa quantità di informazione: nel nostro caso per i modelli CBM 2040 e 8050 ogni settore contiene 256 caratteri (byte).** In Figura 6-6 riportiamo l'organizzazione del settori.

La maggior parte dei dischetti impiegano un eguale numero di settori su ogni traccia sebbene le tracce più esterne siano più lunghe di quelle più interne. I modelli CBM 2040 e 8050 approfittano invece di questa opportunità per scrivere un maggior numero di settori sulle tracce più esterne. Nella Tabella 6-3 sono riportate le tracce per settore: la numerazione delle tracce inizia da 0 per la traccia più esterna e prosegue con numeri maggiori per le tracce più interne.

Se provate a ruotare manualmente un dischetto, all'interno della sua cartuccia di cartone, vedrete comparire un foro in corrispondenza di una finestrella praticata nella cartuccia stessa e posta vicino al centro. Un dischetto con un solo foro si dice che è "settorizzato a software" mentre invece se esistono più fori si dice che è "settorizzato ad hardware". In questo secondo caso sarà presente un foro per ogni settore angolare. I dischetti CBM sono settorizzati a software.

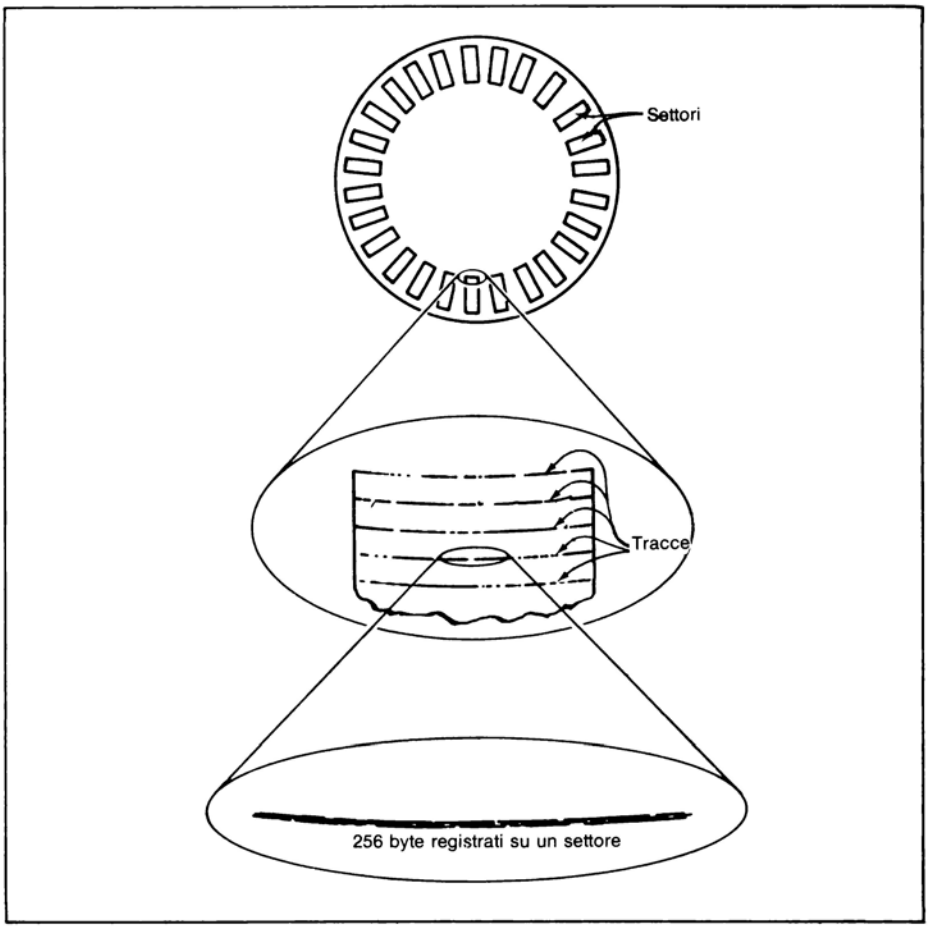


Figura 6-6: Superficie registrata di un dischetto

Directory del dischetto e mappa della disponibilità dei blocchi (BAM)

Due tracce di ogni dischetto sono impiegate come indice del dischetto stesso ("directory").

La directory contiene: il nome che viene assegnato a quel particolare dischetto, i nomi di tutti i file e l'indirizzo dei corispondenti settori di partenza.

La mappa di disponibilità dei blocchi (BAM dalle iniziali di "Block Availability Map") indica quei settori che sono ancora liberi e quelli che sono già stati assegnati ai file.

Se i file fossero invece memorizzati su una cassetta non ci sarebbe alcun bisogno di una directory all'inizio del nastro. Infatti se avessimo per esempio dieci file

Table 6-3. Caratteristiche dei drive

Caratteristiche	Drive 2040		Drive 8050	
Capacità totale	176,640 byte		534,272 byte	
Capacità utile - File sequenziali	170,180 byte		527,812 byte	
Capacità utile - File relativi			182,880 byte	
Tracce	35		77	
Settori per traccia	**Tracce**	**Settori**	**Tracce**	**Settori**
	0-16 17-23 24-29 30-34	21 20(o 19*) 18 17	0-38 39-52 54-65 66-76	29 27 25 23
Byte per settore	256		256	
Blocchi o settori totali	690		2087	
Traccia BAM	18		38	
Traccia Directory	18		39	
* Modello 2				

registrati su una cassetta e volessimo leggere il sesto, la presenza della directory non ci farebbe risparmiare tempo. Siccome i file su cassetta possono avere qualunque lunghezza non c'è alcuna possibilità di convertire il numero del file in una certa posizione sul nastro. L'unico ausilio che avete è quello di avvolgere rapidamente in avanti il nastro fino ad un punto che precede l'inizio del file così da ridurre il tempo di ricerca. Diversamente dovete leggere tutti e cinque i file che precedono il sesto.

Nel caso del dischetto l'accesso ad un file avviene in maniera completamente diversa. Ogni file può essere letto subito in quanto ogni settore è ugualmente e direttamente accessibile. Ovviamente per sapere da quale settore inizia un file è necessario consultare un indice (la directory) che associa appunto al nome di un file l'indirizzo del corrispondente settore iniziale. Nella directory vengono inserite anche altre informazioni quali il tipo di file e la sua lunghezza. Quando apriamo un file su dischetto, il sistema si occupa di leggere subito la directory per sapere da quale settore inizia o potrà inziare il file. Successivamente verranno fatte le operazioni di lettura/scrittura a partire da questo settore iniziale.

Analizziamo ora la struttura dei record di un file su dischetto.

File relativi di dati

In un file relativo tutti i record hanno la stessa lunghezza. È quindi possibile con facilità calcolare in quale settore si trovi un certo record. Supponiamo per esempio che due record di un file relativo occupino esattamente un settore (in pratica ciò è poco probabile, ma ci permette di rendere più comprensibile l'esempio). Il decimo record si troverà quindi nel quinto settore del file.

I file relativi sono disponibili con il software BASIC CBM versione 4.0 e superiori che impiegano il DOS versione 2.0 e superiori.

File sequenziali di dati

I record di un file sequenziale possono avere lunghezze diverse. Non possiamo quindi determinare su quale settore si trovi un certo record appunto perchè ogni record può avere una diversa lunghezza. In questo caso il drive può accedere direttamente al settore di inizio del file, come risulta dalle informazioni contenute nella directory, ma successivamente deve leggere in sequenza tutti i record proprio come un file su cassette. Per esempio non possiamo accedere al decimo record senza prima aver letto tutti i record dall'uno al nove. In Figura 6-7 diamo una rappresentazione concettuale della distribuzione dei record, nei settori, sia per un file relativo che per uno sequenziale.

Tutte le versioni del BASIC CBM prevedono l'impiego dei file sequenziali.

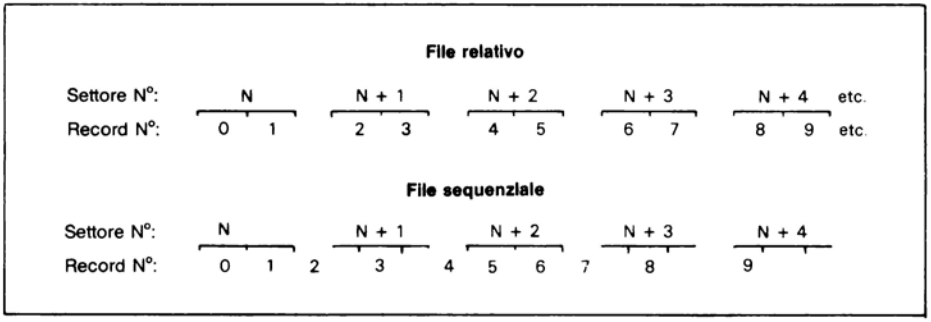


Figura 6-7: Correlazione fra record e settori in un file relativo e in un file sequenziale.

Confronto tra file di dati sequenziali e relativi

Può sorgere spontanea la domanda perchè occuparci di file sequenziali su dischetto dal momento che tali file perdono il grande vantaggio dell'accesso diretto. La risposta è che i file sequenziali registrano le informazioni in maniera molto più densa dei file relativi e quindi sfruttano con maggior rendimento la superficie

magnetica del dischetto. Per meglio chiarire questo concetto facciamo un esempio; prendiamo in considerazione due indirizzi:

Cornelius J. Winkleberger
257631 Avenue of the Americas
Billinghampton
California 92804

Joe R. Smith
5 N St.
York
Iowa 50307

Supponiamo che questi due nominativi con indirizzo facciano parte di un file che rappresenti un elenco di indirizzi. Ogni nominativo con indirizzo costituirà quindi un record. Nel caso di un file relativo ogni record sarà della stessa lunghezza e quindi ogni nominativo/indirizzo occuperà lo stesso spazio sul disco. Per evitare abbreviazioni bisognerà quindi prevedere una lunghezza di record pari al nominativo/indirizzo più lungo. Di conseguenza tutti i nominativi/indirizzi più corti lasceranno dello spazio vuoto e tali vuoti non saranno utilizzabili per nessun impiego!

Se costituiamo invece un file sequenziale ogni nominativo/indirizzo occuperà solamente lo spazio necessario e non avremo quindi alcun spazio inutilizzato.

Il vantaggio maggiore nell'uso dei file relativi è quello per cui potete accedervi e/o modificarli senza alcuna limitazione. Infatti siccome tutti i record hanno una lunghezza fissa è possibile sapere a priori in quale settore si trova ogni singolo record. Di conseguenza possiamo leggere questo settore come anche riscriverlo senza influire sugli altri. Per esempio potete riscrivere il decimo nominativo, con il suo indirizzo, in un file di dati relativo costituito da venti record lasciando inalterati tutti gli altri. Vi è possibile inoltre cancellare un record e se vi è spazio sul disco potete anche aggiungerne dei nuovi.

I file sequenziali dovete trattarli invece come se fossero su cassetta. I loro record devono essere letti sequenzialmente a iniziare dal primo. Potete aggiungere nuovi record in coda al file, ma non potete inserirne dei nuovi in una posizione intermedia. Potete sempre però riscrivere il file e durante questa operazione effettuare le modifiche, cancellazioni o gli inserimenti che desiderate. In conclusione possiamo dire che i file sequenziali fanno un miglior uso della superficie del dischetto, ma sono più complicati da gestire.

Indirizzamento dei settori

I settori, realmente assegnati ad un file di dati su dischetto, non necessitano di essere fisicamente scelti in sequenza continua sulla superificie del disco. Per esempio quando aggiungete dei nuovi record ad un vecchio file potreste supporre che essi vadano a sovrapporsi ad un altro file posto in successione. Questo però non avviene perchè i nuovi record vengono assegnati a settori, posti ovunque, purchè liberi. Quando cancellate dei record il file si contrae e lascia tali settori disponibili

per altri file. **In altre parole si può dire che la logica di gestione del drive prevede che i settori assegnati ad un certo file possano essere ovunque sul dischetto.** Aggiungiamo subito che ciò non presenta problemi, nel caso di file sequenziali, in quanto ogni settore contiene l'indirizzo del settore successivo così che si crea un concatenamento di tutti i settori di un file sequenziale. Nel caso invece di file relativi il problema è più complesso in quanto la logica del drive deve calcolare l'indirizzo di ogni record. Di conseguenza una certa distanza di un record dall'inizio del file deve essere convertita in una distanza o spostamento di settore. Se ritorniamo al nostro esempio di un file, che ha due record per settore e desideriamo accedere al decimo record, ciò equivale ad accedere al quinto settore del file relativo. Poichè i settori non sono sequenziali, è necessario prevedere quindi l'uso di una tabella indice che riporti gli indirizzi reali dei settori. In pratica le cose vanno come riportato qui di seguito:

Numero di record	Numero di settore corrispondente	Indirizzo di traccia e settore	
		Traccia	Settore
1	1	11	4
2	1		
3	2	11	5
4	2		
5	3	11	6
6	3		
7	4	13	9
8	4		
9	5		
10	5	13	10
11	6	9	3

Così per esempio il record numero 6 si trova sul terzo settore che ha indirizzo: traccia 11, settore 6.

Si usa il termine "side sector" per descrivere l'indice dei settori di un file relativo. Attualmente il drive 8050 non può usare tutta la capacità di un dischetto per i file relativi in quanto supererebbe lo spazio consentito per il side sector. Questo è il motivo per cui nella Tabella. 6-3 sono indicati solo 182.880 byte per i file relativi invece della capacità totale di mezzo milione di byte. È previsto che queste limitazioni siano superate nelle prossime versioni del drive 8050.

PROGRAMMAZIONE DEI FILE SU DISCHETTO

A seconda che i file siano sequenziali o relativi è necessario impiegare una diversa tecnica di programmazione. È inoltre previsto che voi possiate effettuare alcune operazioni molto importanti di governo dei dischetti.

Nome dei file su dischetto

I nomi dei file su dischetto seguono le normali regole del BASIC CBM. Normalmente i nomi sono lunghi 16 caratteri e vi consigliamo di non superare mai questo limite.

Le istruzioni DOS identificano un file mediante il suo nome. Potete specificare il nome per intero oppure potete dare solo i primi caratteri seguiti da un asterisco (*). Questo significa che viene scelto il primo file il cui nome soddisfa ai caratteri che avete dato. Ecco alcuni esempi:

Richiesta di file:	PAR*	
File selezionati:	PARITY PARITY,SEC PARITY,N12 PARTITION etc.	Il primo file in cui il nome inizia con PAR

Richiesta di file:	*	
File selezionati:	Il primo file incontrato	

Potete anche ricercare un nome di file specificando alcune lettere, ma non altre. Le lettere che non definite devono essere sostituite da punti interrogativi (?). Ecco alcuni esempi:

Richiesta di file:	N??,SEQ	
File selezionati:	NUM,SEQ NXY,SEQ NAB,SEQ NRA,SEQ etc.	Il primo file con il nome N??, SEQ; con ? qualunque carattere

È possibile infine usare contemporaneamente asterischi e punti interrogativi, come in questo esempio:

Richiesta di file:	NUM??*
File selezionati:	Il primo file con almeno 5 caratteri e i primi tre sono NUM

Versioni del Sistema Operativo a Disco (DOS)

Le istruzioni per la gestione dei dischi nell'ambito del BASIC CBM fanno parte di alcuni programmi che nel loro insieme prendono il nome di Sistema Operativo a Disco (DOS). Non è necessario conoscere molto dei sistemi DOS per poterli usare così come non è necessario essere competenti del funzionamento di un interprete BASIC per poter scrivere programmi in questo linguaggio. **È importante però sapere che esistono varie versioni di DOS CBM in quanto non tutte rispettano le stesse regole.**

Versioni del BASIC CBM

Vi ricordiamo che esistono quattro versioni del BASIC CBM normalmente in uso. La versione BASIC 3.0 e le precedenti sono state fornite con i calcolatori CBM della serie 3000 e 2001. Successivamente si è passati alla versione 4.0.

Le versioni BASIC 1.0, 2.0 e 3.0 sono molto simili. Come abbiamo già avvisato nella prefazione di questo volume le prime versioni le indichiamo generalmente con BASIC < 3.0 mentre la quarta versione è indicata con BASIC 4.0.

I BASIC < 3.0 supportano i file sequenziali e random. Il BASIC 4.0 supporta i file sequenziali, random e relativi.

Il BASIC 4.0 riconosce tutte le istruzioni dei BASIC di versioni precedenti. Quindi se avete un calcolatore con BASIC 4.0 potete usare qualunque istruzione di gestione dei file. Viceversa non è vero il contrario cioè, per esempio, se avete un BASIC 2.0 non potete usare qualunque istruzione del BASIC 4.0.

Il BASIC 4.0 prevede che i drive a disco siano le unità fisiche principali ("default"); se non viene specificata una diversa unità fisica il sistema impone il valore 8. I BASIC < 3.0 assumono invece come unità fisica principale il primo registratore a cassette se non viene data una indicazione diversa.

Sebbene il BASIC 4.0 esegua tutte le istruzioni dei BASIC <3.0 ci sono però alcune incompatibilità, che riguardano le parole di stato o gli errori di file, quando usate la gestione dei file dei BASIC < 3.0 nell'ambito del BASIC 4.0. Per esempio i BASIC <3.0 non prevedono i file relativi; tuttavia se aprite un file usando un BASIC < 3.0 e non specificate il tipo di file, il BASIC 4.0 aprirà invece un file relativo. Un altro inconveniente può succedere quando eseguite una operazione su file con le istruzioni del BASIC < 3.0 e tale operazione è illegale per il BASIC < 3.0, ma è legale per il BASIC 4.0. Allora l'indicatore luminoso di errore del drive diventerà rosso, l'operazione non sarà eseguita ma il BASIC 4.0 riconoscerà invece una parola di stato OK.

APERTURA DEI FILE SU DISCHETTO

In ogni unità a dischetto sono usati dodici buffer di memoria per accedere ai file sia che i dischetti siano inseriti nel primo che nel secondo drive. Appena accedete ad uno di questi file due di questi buffer vengono impegnati per operazioni di tipo superiore. Di conseguenza rimangono dieci buffer disponibili per l'accesso ai file.

Due buffer sono necessari per l'apertura di ogni file sequenziale, mentre tre buffer sono richiesti per l'apertura di ogni file relativo. **Di conseguenza nel BASIC < 3.0 si possono avere al massimo cinque file sequenziali contemporaneamente aperti. Nel caso invece del BASIC 4.0 l'apertura contemporanea dei file dipende dalla combinazione di quanti sono i sequenziali e di quanti sono i relativi.** Le seguenti combinazioni sono possibili:

0 Retativi e 5 Sequenziali
1 Relativo e 3 Sequanziali
2 Relativi e 2 Sequanziali
3 Relativi e 0 Sequanziali

È opportuno precisare però, che sarebbe possibile aumentare il numero dei file aperti collegando altre unità a disco al calcolatore. L'unica limitazione rimane il numero massimo di indirizzi secondari, contemporaneamente usati, che non possono essere più di 13. Ricordiamo che ogni file aperto deve avere un suo specifico indirizzo secondario.

Indirizzi secondari (BASIC < 3.0)

Le versioni BASIC < 3.0 prevedono 16 indirizzi secondari da 0 a 15. Nelle istruzioni OPEN dei BASIC < 3.0 bisogna sempre specificare l'indirizzo secondario. Nel BASIC 4.0 esso viene invece assegnato automaticamente.

Gli indirizzi secondari nei BASIC < 3.0 sono i seguenti:

1. L'indirizzo 0 è usato per caricare i programmi dai dischetti nella memoria del calcolatore.
2. L'indirizzo 1 è usato per salvare i programmi dalla memoria del calcolatore in un file su dischetto.
3. Gli indirizzi da 2 a 14 sono impiegati per accedere ai file di dati. Uno qualunque di questi valori può essere usato purchè non sia già contemporaneamente assegnato in un'altra istruzione OPEN.
4. L'indirizzo 15 è riservato per l'apertura di uno speciale "canale di comando" che permette di accedere alla parola di stato del dischetto oppure di effettuare quelle particolari operazioni, dette di governo, che saranno descritte più avanti in questo capitolo.

Il canale di comando (BASIC < 3.0)

Questo canale è molto importante e richiede una particolare attenzione.

Nel BASIC 4.0 il canale di comando viene automaticamente aperto quando si apre un file. Nei BASIC < 3.0 dovete invece sempre aprirlo separatamente, prima di effettuare ogni operazione sul disco, e lasciarlo aperto sino a che tali operazioni non siano terminate. Vi ricordiamo che attraverso il canale di comando potete interrogare lo stato del dischetto ed effettuare operazioni speciali.

Apertura dei file di dati su dischetto (BASIC 4.0)

Nel BASIC 4.0 l'istruzione di apertura dei file di dati è la DOPEN # (potete usare anche la OPEN dei BASIC < 3.0 perchè è comprensibile dal BASIC 4.0).

L'istruzione DOPEN # deve specificare un numero di file logico e un nome di file. Il collegamento fisico viene stabilito con il drive D0 (default) se non date alcuna indicazione, oppure con il secondo drive D1 se lo indicate espressamente.

Se specificate la lunghezza di un record mediante il parametro LX, il file sarà di tipo relativo. Per effettuare poi operazioni di lettura o di scrittura non dovete indicare alcun parametro nella DOPEN #.

Se non indicate alcuna lunghezza di record il file sarà sequenziale. In tal caso però dovete indicare se eseguirete letture o scritture sul file. Il parametro è W (write) per la scrittura e niente per la lettura.

Il numero di unità fisica è posto eguale a 8 se non aggiungete il parametro ON Uz con indicazione diversa.

Ecco alcuni esempi di DOPEN # in BASIC 4.0:

`10 DOPEN#1,"MAIL"`	Apertura del file logico 1 per accedere al file sequenziale MAIL in lettura. Il dischetto è nel drive 0.
`50 DOPEN#1,"MAIL",D1,W`	Come il precedente, ma in scrittura.
`230 DOPEN#5,"DATALIST",D0 ON U5`	Apertura del file logico 5 per accedere al file sequenziale DATALIST in lettura. Il dischetto è nel drive 0 dell'unità a disco con numero fisico 5.
`100 DOPEN#2,"MAIL",L100`	Apertura del file logico 2 per accedere al file relativo MAIL. Il dischetto è nel drive 0. Accetta sia operazioni di lettura che di scrittura. Se il file è nuovo viene imposta la lunghezza dei record di 100 caratteri (byte). Se esiste già deve avere i record di 100 caratteri (byte).
`25 DOPEN#3,"SAMPLE",L20,D1`	Apertura del file logico 3 per accedere al file relativo SAMPLE sia in lettura che in scrittura. Il dischetto è nel drive 1. Se il file è nuovo deve essere scritto con 20 caratteri (byte) per record, se esiste già deve avere i record lunghi 20 caratteri (byte).

I nomi di file possono essere dati mediante una variabile di stringa invece di una stringa. Per esempio possiamo scrivere:

```
20 S$="SAMPLE"
25 DOPEN#3,S$,L20,D1
```

Apertura di file sequenziali di dati su dischetto (BASIC < 3.0)

Nel BASIC < 3.0 dovete aprire i file su dischetto mediante l'istruzione OPEN. Riportiamo come esempio l'apertura degli stessi file sequenziali indicati più sopra con DOPEN #, ma con l'uso invece di OPEN (ricordiamo che in BASIC < 3.0 non si possono aprire file relativi). Gli indirizzi secondari sono stati scelti a caso.

```
10 OPEN 1,8,2 "MAIL,SEQ"
50 OPEN 1,8,7 "1:MAIL,SEQ,WRITE"
230 OPEN 5,5,3 "0:DATALIST,SEQ"
```

Anche in questo caso la stringa che compare nella OPEN può essere preassegnata mediante una variabile di stringa; per esempio:

```
5 M$="MAIL,SEQ"
10 OPEN 1,8,2,M$
```

Oppure possiamo scrivere con una notazione più complessa:

```
45 M$="MAIL,SEQ"
50 OPEN 1,8,7,"1:"+M$+"WRITE"
```

Errori di apertura dei file

Quando aprite un file si possono verificare degli errori come quelli che qui di seguito riportiamo:

1. Se aprite un nuovo file sequenziale di dati per una operazione di lettura ottenete l'errore FILE NOT FOUND. Infatti un file sequenziale deve già esistere per poter essere letto.
2. Se aprite un vecchio file, ma sbagliate a indicarne il tipo, ottenete l'errore FILE TYPE MISMATCH. Questo si verifica se aprite un vecchio file relativo come sequenziale, oppure se aprite come relativo un vecchio file sequenziale, oppure se aprite un file di programma come file di dati.
3. Non potete aprire un vecchio file sequenziale per operazioni di scrittura. In tal caso ottenere l'errore FILE EXISTS. Potete scrivere solo in un file sequenziale nuovo.

Se dimenticate il nome del file in una OPEN causerete un errore che vi darà un sacco di guai senza averne alcun avviso. Il DOS assumerà che abbiate aperto un nuovo file e se tutto il resto è corretto non avrete infatti alcuna indicazione di errore. Ma come farete a leggere un file a cui non è associato alcun nome?

CHIUSURA DI UN FILE SU DISCHETTO

Per chiudere qualunque file di dati su dischetto in BASIC 4.0 dovete dare l'istruzione:

```
DCLOSE N
```

oppure se siete in BASIC < 3.0, l'istruzione:

```
CLOSE N
```

dove N è il numero di file logico che appare come primo parametro nelle istruzioni DOPEN e OPEN #.

Dovete sempre chiudere un file al termine della sua scrittura altrimenti può succedere che qualche dato sia perso.

Non sarebbe strettamente necessario chiudere un file dopo la sola lettura, ma è una buona precauzione farlo egualmente.

Ricordiamo che tutti i file sono automaticamente chiusi quando viene eseguita l'istruzione END. (In tal caso ovviamente i drive devono essere ancora accesi!) Tuttavia è buona regola chiudere separatamente ogni file con CLOSE e non attendere l'esecuzione dell'ultima istruzione END. Avevamo già parlato di questo a proposito dei file su cassetta e quanto detto vale anche per i file su dischetto.

ERRORI DI DISCHETTO E PAROLA DI STATO DEGLI ERRORI

Su tutti i drive per dischetto della CBM c'è una spia di allarme rossa che svolge la funzione di indicatore di errore. Essa si accende di rosso quando una operazione sul

dischetto è errata e rimane accesa sino a che non venga fatta la opportuna correzione. Ovviamente nel frattempo non può essere eseguita nessuna altra operazione sul dischetto. Per correggere l'errore fermate l'esecuzione del programma premendo il tasto STOP e quindi leggete la parola di stato degli errori.

È consigliabile leggere sempre la parola di stato dopo ogni operazione sul dischetto e introdurre una routine di controllo che ne analizzi il valore assunto. Questa routine dovrebbe sempre fare parte dei programmi di gestione dei dischetti.

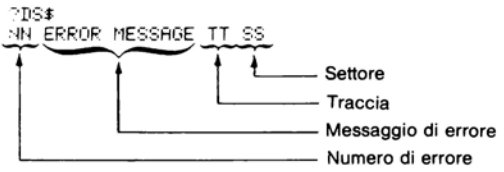
Ricordiamo poi che non potete scrivere su un dischetto se la sua tacca di protezione è coperta. In tal caso si dice che il dischetto è protetto contro la scrittura. Se provate a scrivere su un dischetto protetto, il calcolatore tenterà indefinitivamente di scrivere senza effettuare alcuna registrazione e senza che il drive segnali un errore di stato. In questo caso avrete l'impressione che il calcolatore non faccia niente; penserete di fermarlo con il tasto STOP, ma non vi riuscirete. L'unico modo per venire fuori da tale antipatica situazione è quello di estrarre il dischetto, spegnere il calcolatore e poi riaccenderlo.

Rimozione dello stato di errore dal dischetto

Nell'ambito del BASIC 4.0 potete rimuovere l'errore dal dischetto sia con istruzioni in modo immediato che in modo differito.

Per rimuovere lo stato di errore nel primo caso eseguite l'istruzione PRINT per visualizzare la variabile numerica DS o la stringa DS$.

Nel caso della variabile numerica DS otterrete lo stato come numero decimale la cui interpretazione è data dalla Tabella 6-2. Nel caso invece della stringa DS$ otterrete la visualizzazione di quattro parametri come qui di seguito riportiamo:



I messaggi di errore del dischetto sono riportati nella Appendice B.

Un programma scritto in BASIC 4.0 deve controllare lo stato del dischetto facendo riferimento alla variabile DS come segue:

```
20 IF DS <> 0 THEN PRINT "ERROR"
```

A seguito di un qualunque errore sul dischetto questa istruzione toglie lo stato di errore, ferma l'esecuzione e visualizza il messaggio:

```
ERROR
BREAK IN XXXX
```

XXXX è il numero di linea in cui compare il comando STOP. Se desideriamo avvisare l'operatore di quale tipo di errore si tratta possiamo visualizzare la stringa DS$:

```
20 IF DS <> 0 THEN PRINT DS$:STOP
```

Anche in questo caso, a seguito di un errore su dischetto, questa istruzione toglierà lo stato di errore, fermerà l'esecuzione del programma e visualizzerà i messaggi:

```
NN ERROR MESSAGE TT SS
BREAK IN XXXX
READY
```

dove NN è il numero di errore come riportato nell'Appendice B. TT e SS sono invece il numero di traccia e di settore del settore indirizzato al momento dell'errore.

Errori su dischetto (BASIC < 3.0)

Nell'ambito del BASIC < 3.0 non potete accedere direttamente alle variabili DS e DS$. Per esaminare lo stato di errore dovete aprire un file logico specificando l'unità fisica 8 e l'indirizzo secondario 15. Da questo file dovete quindi richiamare quattro stringhe e visualizzarle. In modo differito potete procedere come segue:

```
10 OPEN 1,8,15
20 INPUT#1,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
40 CLOSE 1
```

Ricordiamo che INPUT # 1 non può essere eseguita in modo immediato.

A$, B$, C$ e D$ contengono la stessa informazione che era data da DS$ nel BASIC 4.0: A$ il numero di errore, B$ il messaggio di errore, C$ il numero di traccia e D$ il numero di settore. Ovviamente A$, B$, C$ e D$ sono nomi di stringa arbitrari e quindi alle righe 20 e 30 si potevano usare altri quattro nomi qualunque.

Quando scrivete un programma in BASIC < 3.0, prima di effettuare qualunque operazione sul dischetto, dovete aprire un file logico con l'indirizzo 8 per l'unità fisica e con l'indirizzo secondario 15. È importante quindi che dopo ogni operazione sul dischetto controlliate lo stato di errore richiamando il numero di errore. Se questo numero è 0 l'operazione è corretta. Ecco un esempio di programmazione:

```
10 OPEN 15,8,15

150 Istruzione di comando del disco
160 REM CONTROLLO STATO DISCHETTO
170 INPUT#15,A$,B$,C$,D$
180 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$: STOP
```

Se nel vostro programma dovete effettuare molte operazioni su dischetto sarà quindi necessario ripetere più volte le linee 170 e 180 dell'esempio di sopra. Di conseguenza potreste pensare di rimandare il controllo ad una subroutine e ciò sarebbe una procedura corretta, ma avreste l'inconveniente di non sapere più quale operazione su dischetto ha causato l'errore. Infatti l'istruzione STOP riporterebbe sempre lo stesso numero di linea della subroutine. Se invece avete la pazienza di ripetere le linee 170 e 180 per ogni operazione su dischetto avrete subito l'indicazione di quale punto del programma ha causato l'errore.

OPERAZIONI DI GOVERNO DEL DISCHETTO

Oltre alle operazioni di lettura e scrittura dei file è previsto che si possano effettuare altre operazioni:

1. Preparare un nuovo dischetto.
2. Cancellare un vecchio dischetto e prepararlo per il riuso.
3. Visualizzare la directory di un dischetto per vedere quali file contiene e quanto spazio libero rimane disponibile.
4. Ricercare quei settori che sono stati assegnati ad un file, ma non ancora usati, così da renderli disponibili per altri file.
5. Copiare un file.
6. Copiare un intero dischetto.
7. Cambiare nome ad un file (solo in BASIC 4.0).
8. Cancellare un file da un dischetto o cambiare il contenuto di un file.

In BASIC < 3.0 ogni file o operazione su disco deve iniziare con una istruzione OPEN. Solo dopo è possibile eseguire operazioni di lettura/scrittura oppure di governo come quelle riportate sopra. Al termine bisognerà dare sempre l'istruzione CLOSE. Nell'ambito del BASIC 4.0 dovete invece dare le istruzioni OPEN e CLOSE solo per le operazioni di lettura/scrittura su file, mentre per le operazioni di governo sono previste delle speciali istruzioni che non richiedono espressamente le OPEN e le CLOSE.

Descriveremo le operazioni di governo prima di entrare nei dettagli della gestione dei file.

Desideriamo anche precisare che le operazioni di governo possono essere eseguite, in modo differito, anche se normalmente sono usate in modo immediato.

Le istruzioni BASIC che impiegheremo per effettuare le operazioni di governo sono dettagliatamente descritte nel capitolo 8. Se a questo punto della trattazione non conoscete bene le istruzioni BASIC saltate allora al capitolo 8 e dopo la sua lettura ritornate a questo punto.

PREPARAZIONE E INIZIALIZZAZIONE DEL DISCHETTO

Un dischetto nuovo non può essere usato così com'è, cioè inserito in un drive e subito scritto. Esso richiede che la sua superificie sia preparata. I settori devono essere segnati sulle tracce e poi devono essere costituite la directory e la mappa di disponibilità dei settori. Da ultimo al dischetto deve essere assegnato un nome. **Potete anche preparare un disco già usato e cioè cancellare tutti i vecchi dati e prepararlo nuovamente per l'uso.**

Comunemente un dischetto viene preparato in modo immediato.

Preparazione del dischetto (BASIC 4.0)

In BASIC 4.0 preparate un dischetto nuovo mediante l'istruzione HEADER come segue:

```
HEADER "NOME DISCO", DX, IYY
```

Dove "NOME DISCO" può essere una qualunque stringa di nome lunga sino a 16 caratteri. YY è l'identificatore che potete assegnare al dischetto e X è il numero del drive in cui avete inserito il dischetto che può essere quindi 0 o 1.

Normalmente la preparazione di un dischetto richiede circa due minuti; se per una ragione qualsiasi essa non può essere effettuata, apparirà allora il seguente avviso:

```
? BAD DISK
```

Le ragioni possibili sono le seguenti:

1. Avete dimenticato di inserire il dischetto nel drive selezionato.
2. Avete sbagliato il numero di drive nell'istruzione HEADER.
3. Vi siete dimenticati di indicare il numero di dischetto nell'istruzione HEADER.
4. Il dischetto è protetto contro la scrittura (cioè la tacca di protezione è coperta).
5. Il dischetto ha la superficie magnetica rovinata.

Quando preparate invece un dischetto già usato è necessario che specifichiate soltanto il numero di drive nella lista parametrica della HEADER. Se indicate un nuovo nome di disco, questo sostituirà il vecchio nome; diversamente rimarrà il vecchio. Se volete potete anche cambiare il numero del disco, ma questo richiede che cambiate, o meglio dire che specifichiate, anche il nome. Cioè non potete inserire nella HEADER solamente il numero YY senza inserire anche il "NOME DISCO". Se non rispettate queste procedure vi verrà dato un errore di sintassi.

Ricordatevi che in BASIC 4.0 i drive per dischetto sono associati all'unità fisica numero 8. Se è necessario però, per qualunque ragione, associare al drive un diverso numero di unità fisica allora dovete aggiungere questa informazione nella lista parametrica della HEADER e cioè porre: ON Uz, dove z è il nuovo numero di unità fisica.

Nel caso di un dischetto già usato la sua preparazione richiede solo pochi secondi.

Ecco alcuni esempi di istruzione HEADER data in modo immediato:

`HEADER "SAMPLE",D0, I01`	Il dischetto nel drive 0 viene preparato con il nome SAMPLE e il numero 01.
`HEADER D0`	Un vecchio dischetto viene preparato nel drive 0 e mantiene il suo nome e il suo numero.
`HEADER "NEW", D1`	Un vecchio dischetto viene preparato nel drive 0. Mantiene il numero, ma prende il nuovo nome NEW.
`HEADER "SAMPLE", D0, I05, ON U7`	Un dischetto viene preparato, con il nome SAMPLE e il numero 05, nel drive 0 con numero fisico 7.
`HEADER D1, I01` `? SYNTAX ERROR`	L'istruzione HEADER non viene eseguita perchè è stato dato il nuovo numero al disco senza un nuovo nome.

Preparazione del dischetto (BASIC < 3.0)

Per preparare un dischetto nell'ambito del BASIC < 3.0 dovete prima aprire il canale di comando del dischetto e quindi eseguire una istruzione PRINT # con lo stesso numero di file logico indicato nella OPEN. La PRINT # deve contenere la seguente stringa racchiusa tra virgolette:

"NEWX: NOME DISCO, YY"

NEW può essere sostituito da N. X è il numero del drive (0 o 1). NOME DISCO è il nome assegnato al dischetto lungo al massimo 16 caratteri. YY è l'identificatore del dischetto.

La OPEN che apre il canale di comando del dischetto può specificare qualunque numero di file logico, ma deve indicare il numero 8 per l'unità fisica e l'indirizzo secondario 15.

Ecco alcuni esempi di inizializzazione in BASIC < 3.0:

`OPEN 1,8,15` `PRINT#1, "N0 SAMPLE,01"`	Un dischetto è inizializzato nel drive 0, con il nome SAMPLE e il numero 01.
`OPEN 3,8,15` `PRINT#3,"NEW1 NEW,01"`	Un dischetto è inizializzato nel drive 1, con il nome NEW e il numero 01.

La preparazione di un dischetto programmata in BASIC < 3.0 non sempre funziona correttamente su un calcoltore CBM che lavori in BASIC 4.0. Qualche volta il drive mantiene in rotazione il dischetto anche dopo che l'inizializzazione è stata terminata.

In BASIC < 3.0 potete preparare un vecchio dischetto e in tal caso tutta la vecchia informazione viene cancellata. Se volete potete mantenere il vecchio identificatore del dischetto non specificando alcun valore per YY nella istruzione PRINT #.

Inizializzazione di un dischetto (BASIC < 3.0)

Quando lavorate in BASIC < 3.0 dovete sempre inizializzare un dischetto, che contenga dei dati, prima di aprire su di esso un file. Per inizializzarlo dovete dare l'istruzione OPEN per aprire il canale di comando ed eseguire quindi una PRINT # con la lettera "I", oppure la parola "INITIALIZE", seguita dal numero di drive. Il numero di drive può essere omesso e in tal caso saranno inizializzati ambedue i dischetti contenuti nei due drive.

I dischetti sono normalmente inizializzati in modo differito.

Quando un dischetto viene inizializzato nessun dato, sulla sua superficie, viene cambiato.

Ecco aluni esempi di inizializzazione in BASIC < 3.0:

```
10 OPEN 1,8,15
20 PRINT#1,"I0"
```

Inizializza il dischetto nel drive 0

```
30 OPEN 3,8,15
40 PRINT#3, "INITIALIZE"
```

Inizializza i dischetti nei drive 0 e 1

Ricordatevi che non è necessario inizializzare un dischetto che è stato appena preparato in quanto la preparazione da luogo contemporaneamente alla inizializzazione.

VISUALIZZAZIONE DELLA DIRECTORY DEL DISCHETTO

Visualizzazione della directory del dischetto (BASIC 4.0)

Prima di accedere ad un dischetto è consigliabile visualizzare sempre la sua directory. Usando il BASIC 4.0 questo è possibile mediante l'istruzione DIRECTORY che normalmente viene data in modo immediato. Qui di seguito vi diamo alcuni esempi:

DIRECTORY	Visualizza le directory dei dischetti nei drive 0 e 1.
DIRECTORY D1	Visualizza la directory del dischetto nel drive 1.
DIRECTORY D1 ON U8	Analoga alla precedente perchè l'unità fisica 8 è il valore "default".
DIRECTORY D0	Visualizza la directory del dischetto nel drive 0.

La parola CATALOG può essere usata invece di DIRECTORY. Il contenuto di una directory viene visualizzato come segue:

```
0       "Nome dischetto"  NNXX

BBBB    "Nome file"       Tipo
BBBB    "Nome file"       Tipo
.
.
.
YYYY    BLOCK FREE
```

Il nome del dischetto e il numero di identificazione appaiono all'inizio con i caratteri in campo inverito. NN è il numero di dischetto e XX è la versione di DOS. Successivamente vengono indicati i file con i loro nomi, il numero BBBB di blocchi (settori) assegnati e il tipo di file: REL per relativo, SEQ per sequenziale e PRG per i programmi. Al termine viene dato il numero YYYY di blocchi ancora liberi. (Vi possono essere anche file di utente come quelli descritti nel capitolo 7).

Quando date il comando DIRECTORY dovete avere prima, ovviamente, inserito i dischetti nei drive. Un errore molto comune è quello di battere DIRECTORY semplicemente quando si vuole la directory di un dischetto inserito nel drive 0. Se non vi è alcun dischetto nel drive 1 allora l'indicatore luminoso diverrà rosso e non apparirà alcuna directory. Ricordatevi che in tal caso dovete rimuovere l'indicazione di errore leggendo lo stato del dischetto (battete ?DS$ < CR >). Non potete riusare il drive fino a che l'indicatore luminoso segnala errore. Otterrete sempre una indicazione di errore se indicherete il drive sbagliato nell'istruzione DIRECTORY. Per esempio se il dischetto è presente nel drive 1 e voi date in modo immediato il comando:

```
DIRECTORY D0
```

vi verrà segnalato errore e non sarà visualizzata la directory.

Visualizzazione della directory del dischetto (BASIC < 3.0)

In BASIC <3.0 ottenete la directory mediante il comando LOAD come segue:

```
LOAD "$X", Y
```

dove X è il numero del drive (0 o 1) e Y è il numero dell'unità fisica (generalmente 8). Qui di seguito vi diamo un esempio standard di come caricare e poi listare una directory:

```
LOAD "$0", 8
SEARCHING FOR $0
LOADING
READY
LIST
```

RIORDINO DI UN DISCHETTO (COLLECT)

Nel BASIC 4.0 esiste il comando COLLECT che permette di "mettere ordine" in un dischetto.

Questo comando identifica i settori che sono stati assegnati ai file, ma non ancora utilizzati e, come dice il nome, ne fa una colletta e li rende nuovamente disponibili. Di conseguenza la directory viene modificata e aggiornata.

Il comando COLLECT viene normalmente dato in modo immediato come negli esempi che seguono:

COLLECT	*Riordino dei dischetti nei due drive*
COLLECT D0	*Riordino del dischetto nel drive 0*

In alcune versioni del BASIC 4.0 vi sono delle difficoltà ad eseguire l'istruzione SCRATCH di un file, cioè il file non viene cancellato, se prima non è stato correttamente chiuso. Se il vostro calcolatore CBM presenta questo problema è sufficiente che voi eseguiate prima un comando COLLECT e poi quello SCRATCH. In questo caso, anche se il file non aveva una chiusura regolare, verrà regolarmente cancellato.

COPIA DI FILE E DISCHETTI

È consigliabile che voi teniate sempre una copia di ogni file che ritenete importante. Ovviamente tale copia non dovrà essere sullo stesso dischetto in quanto tutto il dischetto potrebbe essere accidentalmente rovinato.

Nel BASIC CBM sono previste delle istruzioni che permettono di copiare sia un singolo file come anche un intero dischetto.

Copia di file (BASIC 4.0)

In BASIC 4.0 l'istruzione COPY permette di copiare un singolo file o un intero dischetto. Questa istruzione però fa riferimento ad una unica unità fisica per cui le copie devono avvenire su uno stesso dischetto oppure da un drive all'altro di una stessa unità a disco.

Se nella lista parametrica dell'istruzione COPY non specificate alcun nome di file allora tutti i file saranno ricopiati. Se invece indicate il nome di un file, solo quello sarà ricopiato. Qui di seguito vi diamo alcuni esempi:

`COPY D0 TO D1`	Copia tutti i file del dischetto nel drive 0 sul dischetto nel drive 1.
`COPY D0, "TESTDATA" TO D1, "TESTDATA"`	Copia il file TEST DATA del dischetto nel drive 0 sul dischetto nel drive 1 con lo stesso nome.
`COPY D1, "TESTDATA" TO D0, "NEWTEST"`	Copia il file TEST DATA del dischetto nel drive 1 sul dischetto nel drive 0 e cambia il nome in NEW TEST.

Copia di file (BASIC < 3.0)

Se lavorate in BASIC < 3.0 l'istruzione che vi permette di copiare un file è la PRINT# seguita dalla stringa parametrica:

"COPY M: nome nuovo = N: nome vecchio"

Invece di COPY potete indicare solo la lettera C. Nella stringa ritroviamo: nome vecchio che è il nome del file da copiare (detto anche sorgente); N che è il numero del drive corrispondente; nome nuovo che è il nome del file duplicato che può essere eguale, oppure no, al vecchio (detto anche destinazione) e infine M che è il numero del drive che corrisponde al nuovo file.

Ecco aluni esempi:

```
OPEN 15,8,15
PRINT#15, "COPY1:MAILDATA=0:MAILDATA"
CLOSE 15
```

Copia il file MAILDATA del dischetto nel drive 0 sul dischetto drive 1 e mantiene il nome.

```
OPEN 15,8,15
PRINT#15, "CO:NEWTEST=1:TESTDATA"
CLOSE 15
```

Copia il file TESTDATA del dischetto nel drive 1 sul dischetto nel drive 0 e cambia il nome in NEWTEST.

Concatenamento di file (BASIC < 3.0)

Durante l'esecuzione della copia di un file, mediante l'istruzione PRINT # del BASIC **< 3.0, è possibile concatenare due, tre o quattro file sorgenti in un unico file destinazione.** Per spiegare questo concatenamento facciamo un esempio in modo immediato in cui desideriamo unire due file, di nome DATA 1 e DATA 2 posti su un dischetto inserito nel drive 0, in un unico file di nome DATA X su un dischetto inserito invece nel drive 1. Ecco l'esempio:

```
OPEN 15,8,15
PRINT#15,"C1:DATAX=1:DATA1,1:DATA2"
CLOSE 15
```

I file sorgenti che si desidera concatenare possono provenire da uno stesso dischetto oppure da dischetti diversi: è sufficiente indicare da quali drive provengono nella lista parametrica della PRINT #.

Errori nella copia dei file

Una operazione di copia non può indicare, come nome per il file destinazione, un nome che già esiste. Se si commette questo errore la copia non sarà fatta. Di conseguenza si otterrà l'accensione della spia di errore rossa del drive e se cercherete di controllare lo stato di errore vi verrà risposto che il file esiste già (FILE EXISTS).

Quando copiate tutti i file di un dischetto su un altro con l'istruzione COPY del BASIC 4.0, un nome sorgente non deve essere già presente sul dischetto destinazione. Se questo avviene l'operazione di copia si ferma immediatamente e la luce rossa di errore si accende. Anche i file, i cui nomi compaiono dopo quello rifiutato, non saranno copiati.

Duplicazione di un dischetto

Se desiderate copiare un dischetto potete copiare tutti i file in esso contenuti oppure fare un duplicato ("backup") dell'intero dischetto. Potrebbe sembrare la stessa cosa, ma in realtà i due metodi sono alquanto diversi. **Nel secondo caso la duplicazione o "backup" da luogo ad una copia esatta del dischetto sorgente cioè con lo stesso nome,**

lo stesso numero, la stessa directory e gli stessi file. Nel primo caso invece la copia dei singoli file comporta che il nome del dischetto destinazione rimanga invariato, come anche il suo numero e tutti i file che già conteneva continuino a rimanere presenti. In altre parole, in quest'ultimo caso, al dischetto destinazione si sono semplicemente aggiunti i file del dischetto sorgente.

"Backup" di un dischetto (BASIC 4.0)

La **duplicazione completa di un dischetto in BASIC 4.0 si esegue mediante l'istruzione BACKUP.** È possibile eseguire la duplicazione tra il drive 0 e 1 e viceversa tra il drive 1 e 0 con l'eventuale indicazione anche dell'unità fisica. Ecco alcuni esempi:

```
BACKUP D0 TO D1
```
Fa la copia del dischetto nel drive 0 sul dischetto nel drive 1.

```
BACKUP D1 TO D0 ON U5
```
Fa la copia del dischetto nel drive 1 sul dischetto nel drive 0. L'unità a disco è indirizzata come unità fisica 5.

Con l'istruzione BACKUP si può ricopiare su un dischetto che non sia stato preparato in precedenza. Se però è ritenuta necessaria, l'operazione di preparazione del dischetto destinazione deve essere effettuata prima che inizi quella di BACKUP.

Duplicazione di un dischetto (BASIC < 3.0)

L'istruzione per duplicare un dischetto in BASIC < 3.0 è PRINT # seguita dalla stringa:

"DUPLICATEN = M"

Potete copiare dal drive 0 al drive 1 o viceversa purchè i drive siano di una stessa unità fisica. Anche in questo caso invece di tutta la parola DUPLICATE potete usare la sola lettere D. N è il numero del drive di destinazione mentre M è quello della sorgente.

Ecco un esempio che potete dare in modo immediato:

```
OPEN 15,8,15
PRINT#15, "D1=0"
CLOSE 15
```
Fa la copia del dischetto nel drive 0 sul dischetto nel drive 1.

CAMBIAMENTO DEL NOME DI UN FILE

È possibile cambiare il nome sia di programmi che di file di dati. Più frequentemente si cambia il nome ai programmi durante la loro stesura o correzione.

Cambiare il nome di un file (BASIC 4.0)

L'istruzione per cambiare il nome di un file in BASIC 4.0 è RENAME come potete vedere nell'esempio che segue:

```
RENAME D0, "SEQ.NUM.B4" TO "SEQNUM"
```

Cambiare il nome di un file (BASIC < 3.0)

Per cambiare il nome di un file in BASIC < 3.0 è necessario invece usare l'istruzione PRINT # seguita dalla stringa:

"RENAMEX: nome nuovo = nome vecchio"

Anche in questo caso invece dell'intera parola RENAME si può dare la sola lettera iniziale R. X è il numero del drive che contiene il dischetto su cui è registrato il file; nome nuovo è il nome che sostituirà il nome vecchio. Per esempio:

```
OPEN 15,8,15                          Il file cambia nome da
PRINT#15, "R0:SEQNUM=SEQ.NUM.B4"      SEQ.NUM.B4 in SEQNUM
CLOSE 15
```

CANCELLAZIONE DI FILE

Qualunque file può essere cancellato da un dischetto. Se lo fate in modo immediato il calcolatore vi chiederà la conferma ARE YOU SURE? (cioè siete sicuri di volerlo cancellare) a cui dovete rispondere "YES" seguito dal ritorno del carrello. Se non date la conferma il file non sarà cancellato. Se lavorate invece in modo differito non vi verrà chiesta nessuna conferma.

Cancellazione di un file (BASIC 4.0)

Se usate il BASIC 4.0 il comando per cancellare un file è SCRATCH. Per esempio in modo immediato potete dare:

```
SCRATCH D0, "REL.NUM.B4"        Cancella il file REL.NUM.B4 nel drive D0
```

In modo differito invece:

```
.
.
240 DCLOSE
250 SCRATCH D0, "REL.NUM.B4"
.
.
```

In alcune versioni di BASIC 4.0 vi possono essere dei problemi per cancellare un

file che non è stato correttamente chiuso. Se vi trovate davanti a queste difficoltà dovete dare prima il comando COLLECT e poi SCRATCH.

Cancellazione di un file (BASIC < 3.0)

In BASIC < 3.0 potete cancellare uno o più file con una unica istruzione PRINT # seguita dalla stringa:

"SCRATCH: nome file"

Precisiamo che invece di SCRATCH potete dare solo S; X è il numero del drive interessato. Ecco un esempio in modo immediato:

```
OPEN 15,8,15
PRINT#15, "S0:REL.NUM.B4"
CLOSE 15
```

Cancella il file REL.NUM.B4 nel drive D0

Se dovete cancellare due o più file basterà che aggiungiate, nella stringa parametrica, gli altri numeri di drive e nomi di file. Per maggiore chiarezza guardate come viene modificato l'esempio precedente per cancellare due file:

```
OPEN 15,8,15
PRINT#15, "S0:REL.NUM.B4,1:REL.NUM.B<3"
CLOSE 15
```

Cancella REL.NUM.B4 (D0) e REL. NUM.B < 3 (D1)

Esiste un altro metodo per cancellare più file che abbiano nomi analoghi. **Se nel nome del file ponete un asterisco dopo alcune lettere saranno cancellati tutti i file i cui nomi iniziano con le stesse lettere.** Per esempio:

```
OPEN 15,8,15
PRINT#15, "S0:NUM*"
CLOSE 15
```

Tutti i file nel drive 0 o i cui nomi iniziano con le lettere NUM saranno cancellati.

Se invece sostituite una o più lettere con un punto interrogativo saranno cancellati tutti i file che abbiano una qualunque lettera al posto dei punti interrogativi. Per esempio vi mostriamo come si possono cancellare tutti i file i cui nomi iniziano con NUM, terminano con .SEQ e hanno quattro caratteri nel mezzo:

```
OPEN 15,8,15
PRINT#15, "S0:NUM????.SEQ"
CLOSE 15
```

Anche in BASIC 4.0, mediante il comando SCRATCH, è possibile cancellare più file con la medesima logica (uso dell'asterisco e/o del punto interrogativo).

Sostituizione di un file (BASIC < 3.0)

Sebbene in BASIC < 3.0 non sia possibile scrivere in un vecchio file è previsto però che se ne possa cambiare il contenuto. Per effettuare questa operazione il vecchio file deve essere aperto per la scrittura con un carattere @ che appaia come primo arattere nella stringa della lista parametrica. Per esempio il file MAIL, che compare qui di seguito, può essre un vecchio file:

```
50 OPEN 1,8,7,"@1:MAIL,SEQ,WRITE"
```

FILE SEQUENZIALE DI DATI

Sia il BASIC 4.0 che i BASIC < 3.0 prevedono l'uso di file sequenziali.

Un file sequenziale deve essere aperto però solo per operazioni di lettura oppure di scrittura, ma mai per tutte e due contemporaneamente. Un file sequenziale nuovo deve essere aperto in scrittura e in tale momento esso viene anche creato; un file nuovo non può mai essere aperto in lettura. Un file già esistente può essere aperto solo per operazioni di lettura.

SEPARATORI DI CAMPO IN FILE SEQUENZIALI

In un file sequenziale di dati è preferibile che le variabili numeriche siano separate da un carattere di ritorno del carrello mentre quelle di stringa possono terminare sia con il carattere virgola sia con quello di ritorno del carrello.

Noi raccomandiamo di usare sempre però il carattere di ritorno del carrello per separare i campi. Se usate invece la virgola per separare le stringhe non avrete un particolare vantaggio e potreste incorrere in qualche difficoltà di programmazione.

Se terminate tutti i campi con il ritorno del carrello allora le regole per scrivere file sequenziali sono molto semplici: usate l'istruzione PRINT # come se fosse invece una semplice PRINT che debba visualizzare le variabili in colonna verticale. I dati saranno poi letti dal file mediante la INPUT # o la GET #. Se lavorate con il BASIC 4.0 e con il DOS 2.0 l'istruzione PRINT # aggiungerà automaticamente il carattere di ritorno del carrello al termine di una linea se il numero di file logico è uguale o superiore a 128. Se è minore invece non viene aggiunto niente.

SCRITTURA DI DATI NUMERICI IN UN FILE SEQUENZIALE

Vi proponiamo ora un semplice esempio di file sequenziale numerico costituito da dieci record e ogni record composto da dieci numeri:

Record 1:	1	2	3	4	5	6	7	8	9	10
Record 2:	101	102	103	104	105	106	107	108	109	110
Record 3:	201	202	203	204	205	206	207	208	209	210
Record 4:	301	302	303	304	305	306	307	308	309	310
Record 5:	401	402	403	404	405	406	407	408	409	410
etc.										

Il programma provvederà anche alla lettura e visualizzazione del file.

Qui di seguito vi diamo il listing del programma sia nella versione per BASIC 4.0 (programma "SEQ. NUM. B4") sia in quella per BASIC < 3.0 (programma "SEQ. NUM. B < 3").

Versione BASIC 4.0

```
10 REM PROGRAMMA "SEQ.NUM.B4"
20 DOPEN#1,"TESTDATA",W
30 IF DS<>0 THEN PRINT DS$:STOP
40 REM SCRIVE 10 RECORD
50 FOR R=1 TO 10
60 REM SCRIVE 10 CAMPI PER RECORD
70 FOR C=1 TO 10
80 PRINT#1,(R-1)*100+C
85 IF DS<>0 THEN PRINT DS$:STOP
90 NEXT C
100 NEXT R
110 DCLOSE#1
200 REM LEGGE IL FILE E LO VISUALIZZA
210 DOPEN#1,"TESTDATA"
215 IF DS<>0 THEN PRINT DS$: STOP
220 FOR R=1 TO 10
230 PRINT "RECORD";R;
240 REM LEGGE E VISUALIZ. RECORD SUCCESSIVO
250 FOR C=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$: STOP
270 PRINT N;
280 NEXT C
290 PRINT
300 NEXT R
310 DCLOSE#1
320 SCRATCH D0,"TESTDATA"
330 STOP
```

Versione BASIC < 3.0

```
10 REM PROGRAMMA "SEQ.NUM.B<3"
20 OPEN 15,8,15: REM CANALE DI COMANDO
21 INPUT#15,A$,B$,C$,D$
22 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
23 PRINT#15, "I0"
24 OPEN 1,8,2,"0:TESDATA<3,SEQ,W": REM FILE DI
30 INPUT#15,A$,B$,C$,D$
31 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
40 REM SCRIVE 10 RECORD
50 FOR R=1 TO 10
60 REM SCRIVE 10 CAMPI PER RECORD
70 FOR C=1 TO 10
80 PRINT#1,(R-1)*100+C
85 INPUT#15,A$,B$,C$,D$
86 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
90 NEXT C
100 NEXT R
110 CLOSE 1
120 CLOSE 15
200 REM LEGGE IL FILE E LO VISUALIZZA
210 OPEN 15,8,15: REM CANALE DI COMANDO
211 INPUT#15,A$,B$,C$,D$
212 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
213 OPEN 1,8,2,"0:TESDATA<3,SEQ,": REM FILE DATI
215 INPUT#15,A$,B$,C$,D$
216 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
220 FOR R=1 TO 10
```

```
230 PRINT "RECORD";R;
240 REM LEGGE E VISUALIZ. RECORD SUCCESSIVO
250 FOR C=1 TO 10
260 INPUT#1,N
265 INPUT#15,A$,B$,C$,D$
266 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
270 PRINT N;
280 NEXT C
290 PRINT
300 NEXT R
310 CLOSE 1
320 SCRATCH D0,"TESTDATA<3"
330 CLOSE 15
340 STOP
```

Caricate quindi il programma nella versione che compete al vostro calcolatore, memorizzatelo e ponetelo in esecuzione. Sul display dovrà allora apparirvi un testo come quello che segue:

```
RECORD 1  1   2   3   4   5   6   7   8   9   10
RECORD 2  101  102  103  104  105  106  107  108  109  110
RECORD 3  201  202  203  204  205  206  207  208  209  210
RECORD 4  301  302  303  304  305  306  307  308  309  310
RECORD 5  401  402  403  404  405  406  407  408  409  410
RECORD 6  501  502  503  504  505  506  507  508  509  510
RECORD 7  601  602  603  604  605  606  607  608  609  610
RECORD 8  701  702  703  704  705  706  707  708  709  710
RECORD 9  801  802  803  804  805  806  807  808  809  810
RECORD 10  901  902  903  904  905  906  907  908  909  910
```

Vediamo ora di comprendere la struttura logica del programma.

Le istruzioni dalla riga 10 alla 120 creano il file sequenziale di dati e scrivono i suoi dieci record. Le istruzioni dalla riga 200 alla 320 leggono il contenuto del file, record per record, e ne visualizzano il contenuto.

Guardiamo come i file sono stati aperti e chiusi. Nella versione in BASIC 4.0 il file TESTDATA è stato aperto in scrittura dall'istruzione DOPEN # della riga 20 con il numero 1 di file logico. Il file viene quindi chiuso alla riga 110 prima di essere riaperto in lettura con la DOPEN # alla riga 210. Durante la seconda apertura viene utilizzato lo stesso numero 1 di file logico anche se non è assolutamente necessario mantenere il medesimo numero. Alla riga 310 il file viene definitivamente chiuso.

Nel programma in versione BASIC < 3.0 il file TESTDATA < 3 viene aperto in scrittura con l'istruzione OPEN alla riga 24 con numero di file logico 1 e indirizzo secondatio 2. Viene quindi chiuso alla riga 110 prima di essere riaperto in lettura con la OPEN della riga 213. TESTDATA < 3 viene chiuso poi definitivamente alla riga 310. Il programma in BASIC < 3.0 apre anche il canale di comando con l'istruzione OPEN alla riga 20 con il numero di file logico 15, che potrebbe essere anche diverso, e l'indirizzo secondario 15 che invece è obbligatorio. È consuetudine usare il numero 15 di file logico per il canale di comando, nei programmi in BASIC < 3.0, per associarlo allo steso numero del canale secondario. Il canale di comando viene chiuso alla riga 120, successivamente è riaperto alla riga 210 ed è poi definitivamente chiuso alla riga 330. Non sarebbe strettamente necessario chiudere

e poi riaprire il canale di comando, cioè le righe 120 e 210 potrebbero essere eliminate, ma questo permette di separare il programma in due moduli che potrebbero essere eseguiti indipendentemente.

Desideriamo farvi notare che il programma in BASIC < 3.0 inizializza il dischetto alla riga 23, ma questa operazione dovrebbe essere ripetuta dopo la riapertura del canale di comando, alla riga 210, se le due parti del programma fossero realmente eseguite separatamente.

Ambedue le versioni di questo programma cancellano alla fine il file di dati (riga 320). Se i file di dati non fossero cancellati non sarebbe possibile eseguire una seconda volta il programma. Provate infatti ad eliminare l'istruzione 320 ed eseguire due volte il programma. Alla seconda esecuzione otterrete il messaggio di errore FILE ALREADY EXISTS al momento dell'apertura in scrittura del file (alla riga 20 nella versione in BASIC 4.0 ed alla riga 24 nella versione in BASIC < 3.0).

I file temporanei di dati devono essere sempre cancellati alla fine di un programma se non è necessario conservarli. Infatti se tali file temporanei di dati non vengono cancellati, non sarebbe possibile riutilizzarli in caso di riesecuzione del programma.

Esaminiamo ora la logica di controllo dello stato del dischetto nei nostri due programmi. Questa parte di programmazione viene spesso tralasciata da programmatori "frettolosi". (Le istruzioni, in BASIC 4.0 e in BASIC < 3.0, necessarie per il controllo dello stato del dischetto sono già state descritte in questo capitolo).

Il programma in BASIC 4.0 controlla lo stato del dischetto alle righe 30, 85, 215 e 265. In ognuno di questi casi se lo stato non è 0, la stringa DS$ viene visualizzata per identificare l'errore e l'esecuzione del programma viene fermata.

Il programma in BASIC < 3.0 segue la stessa logica introducendo lo stato per mezzo delle variabili di stringa A$, B$, C$ e D$. Le quattro stringhe vengono visualizzate se il valore numerico di A$ non è 0. Le istruzioni di controllo dello stato sono alle righe 21 e 22, 30 e 31, 85 e 86, 210 e 211, 215 e 216, 265 e 266.

I due programmi contengono poi le stesse istruzioni per scrivere e leggere i record e per visualizzare i dati.

Le istruzioni dalla riga 50 alla 100 effettuano la scrittura dei record nel file. In particolare il ciclo FOR NEXT esterno, il cui indice è R, conta i record mentre il ciclo FOR NEXT interno con indice C conta i campi in ogni record. L'istruzione PRINT # alla riga 80 scrive ogni singolo campo del file. Per quanto riguarda il carattere di ritorno del carrello non è necessario batterlo in quanto vi è solo una variabile nella lista della PRINT #. Vi ricordiamo infatti che la scrittura dei campi è corretta se, sostituendo la PRINT # con una PRINT, si ottiene una visualizzazione dei campi in colonna verticale.

Le istruzioni dalla riga 220 alla 300 leggono i dati dal file e li visualizzano. Il ciclo esterno FOR NEXT con indice R legge i record; l'istruzione PRINT alla riga 230 inizia la visualizzazione di ogni record con il numero progressivo del record. Il ciclo interno FOR NEXT, con indice C, legge un campo alla volta con la INPUT # della riga 260. I campi che appartengono ad ogni record sono visualizzati con l'istruzione

PRINT della riga 270. L'istruzione PRINT della riga 290 forza un ritorno del carrello al termine della visualizzazione di ogni record.

È importante a questo punto chiarire un concetto. **Sebbene si sia detto che il file sequenziale dell'esempio è costituito da dieci record con dieci campi, sulla superficie del dischetto il file è composto invece da soli campi (100) separati dal carattere di ritorno del carrello.** Se noi volessimo "vedere" i dati così come sono registrati sul dischetto non troveremmo niente che distingua un record da un altro. È la struttura logica del programma che tiene conto di come i campi sono ripartiti in record.

Per dimostrarvi quanto abbiamo appena detto, provate a cambiare la seconda parte dei programmi così da avere 12 record con 8 campi ciascuno. In altre parole cambiate le righe 220 e 250 come segue:

```
220 FOR R=1 TO 12

250 FOR C=1 TO 8
```

Dopo questi cambiamenti ponete in esecuzione il vostro programma; otterrete allora:

```
RECORD 1   1   2   3   4   5   6   7   8
RECORD 2   9  10  101  102  103  104  105  106
RECORD 3  107  108  109  110  201  202  203  204
RECORD 4  205  206  207  208  209  210  301  302
RECORD 5  303  304  305  306  307  308  309  310
RECORD 6  401  402  403  404  405  406  407  408
RECORD 7  409  410  501  502  503  504  505  506
RECORD 8  507  508  509  510  601  602  603  604
RECORD 9  605  606  607  608  609  610  701  702
RECORD 10  703  704  705  706  707  708  709  710
RECORD 11  801  802  803  804  805  806  807  808
RECORD 12  809  810  901  902  903  904  905  906
```

Ogni record inizia dove il precedente termina. Si vede quindi che non vi è alcun legame tra la struttura dei campi/record in scrittura e quella in lettura.

Se dovete scrivere due o più variabili numeriche in un file mediante una unica istruzione PRINT #, dovete allora forzare i caratteri di ritorno del carrello. Questo è possibile mediante la funzione CHR$. Supponiamo di modificare la riga 80 come segue:

```
80 PRINT#1,R,CHR$(13),C,CHR$(13),(R-1)*100+C
```

Normalmente è preferibile assegnare il valore di CHR$ ad una stringa e poi inserire quest'ultima nella PRINT #:

```
15 C$=CHR$(13)
.
.
80 PRINT#1,R,C$,C,C$,(R-1)*100+C
```

Con la modifica fatta alla riga 80 ci sono adesso 30 numeri in ogni record invece di 10. Nella seconda parte del programma è necessario quindi leggere e visualizzare 30 numeri per ogni record. Un modo semplice (ma poco elegante) per visualizzare questi 30 valori è quello di estendere il limite superiore per l'indice C nell'istruzione FOR della riga 250:

```
250 FOR C=1 TO 30
```

Inserite allora queste modifiche nel vostro programma e verificate che effettivamente il file di dati sequenziale sia stato scritto come vi abbiamo indicato.

SCRITTUA DI STRINGHE IN UN FILE SEQUENZIALE

Come abbiamo già detto le variabili di stringa possono essere separate sia da un carattere virgola oppure da un carattere di ritorno del carrello. Tuttavia nel caso di file sequenziali la virgola non porta nessun particolare vantaggio. Per questo motivo preferiamo usare sempre il carattere di ritorno del carrello come elemento di separazione delle stringhe.

Per quanto riguarda poi la struttura logica di un programma, che scrive un file sequenziale di stringhe, non vi è nessuna particolare differenza rispetto a quella di un programma che scrive invece un file numerico.

Per illustrare come si possa procedere in questo caso scriviamo un programma per la gestione di un archivio di indirizzi ("mailing list"). Qui di seguito vi diamo sia la versione in BASIC 4.0 che in BASIC < 3.0 e una prova di esecuzione.

Versione BASIC 4.0

```
10 REM PROGRAMMA "SEQ.MAIL.B4"
15 REM
20 REM QUESTO PROGRAMMA ILLUSTRA L'USO DI FILE
21 REM DI STRINGHE SU DISCHETTO
25 REM
30 DATA "   NOME: "," VIA: "," CITTA': "," STATO: "," CAP: "
40 DOPEN#1,"SEQ.MAILDATA",W
50 IF DS<>0 THEN PRINT DS$: STOP
60 PRINT"□ BATTERE NOME E INDIRIZZO:□□"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT"BATTERE Y PER REGISTRARE; N PER RIPETERE I DATI";
130 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM SCRIVE NOME E INDIR. NEL FILE
160 FOR I=1 TO 5
170 PRINT#1,AD$(I)
180 NEXT I
190 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
200 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="Y" THEN 60
220 DCLOSE#1
```

```
300 REM VISUALIZZ. NOMI E INDIRIZZI
310 DOPEN#1,"SEQ.MAILDATA"
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM PULISCE LO SCHERMO E VISUALIZZA I DATI
350 PRINT"□NN"
360 RESTORE
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390 INPUT#1,AD$
400 IF DS<>0 THEN PRINT DS$: STOP
410 PRINT AD$
420 NEXT I
430 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
440 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 440
450 IF Y$="Y" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"SEQ.MAILDATA"
480 STOP
```

Versione BASIC < 3.0

```
10 REM PROGRAMMA "SEQ.MAIL.B<3"
15 REM
20 REM QUESTO PROGRAMMA ILLUSTRA L'USO DI FILE
21 REM DI STRINGHE SU DISCHETTO
25 REM
30 DATA "   NOME: ","    VIA: "," CITTA': ","  STATO: ","    CAP: "
40 OPEN 15,8,15:  REM CANALE DI COMANDO
41 INPUT#15,A$,B$,C$,D$
42 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
43 PRINT#15,"I0"
44 OPEN 1,8,2,"0:MAILDATA<3,SEQ,W"
50 INPUT#15,A$,B$,C$,D$
51 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
60 PRINT"□ BATTERE NOME E INDIRIZZO:NN"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT"BATTERE Y PER REGISTRARE; N PER RIPETERE I DATI";
130 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM SCRIVE NOME E INDIR. NEL FILE
160 FOR I=1 TO 5
170 PRINT#1,AD$(I)
180 NEXT I
190 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
200 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="Y" THEN 60
220 CLOSE 1
300 REM VISUALIZZ. NOMI E INDIRIZZI
310 OPEN 1,8,2,"0:MAILDATA<3,SEQ"
320 INPUT#15,A$,B$,C$,D$
321 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM PULISCE LO SCHERMO E VISUALIZZA I DATI
350 PRINT"□NN"
360 RESTORE
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390 INPUT#1,AD$
400 INPUT#15,A$,B$,C$,D$
401 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
410 PRINT AD$
420 NEXT I
```

```
 430 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
 440 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 440
 450 IF Y$="Y" THEN 350
 460 CLOSE 1
 470 SCRATCH D0,"SEQ.MAILDATA"
 480 STOP
READY.
```

```
  BATTERE NOME E INDIRIZZO:


     NOME: CLAUDIO MONET
      VIA: CAMPI ELISI 100
    CITTA': PARIGI
    STATO: FRANCIA
      CAP: F-0000
BATTERE Y PER REGISTRARE; N PER RIPETERE I DATIY
BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINAREY
  BATTERE NOME E INDIRIZZO:


     NOME: EDMONDO HALLEY
      VIA: GREENWICH
    CITTA': LONDRA
    STATO: GRAN BRETAGNA
      CAP: UK-0000
BATTERE Y PER REGISTRARE; N PER RIPETERE I DATIY
BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINAREY
  BATTERE NOME E INDIRIZZO:


     NOME: ENRICO FERMI
      VIA: PANISPERNA
    CITTA': ROMA
    STATO: ITALIA
      CAP: 00100
BATTERE Y PER REGISTRARE; N PER RIPETERE I DATIY
BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINAREY
  BATTERE NOME E INDIRIZZO:


     NOME: GALILEO FERRARIS
      VIA: VALENTINO
    CITTA': TORINO
    STATO: ITALIA
      CAP: 10100
BATTERE Y PER REGISTRARE; N PER RIPETERE I DATIY
BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINAREN
```

Esaminiamo ora la logica del nostro programma.

Le istruzioni dalla riga 40 alla 220 caricano nomi e indirizzi dalla tastiera e li trasferiscono sul file sequenziale. Le istruzioni dalla riga 300 alla 460 leggono i nomi e gli indirizzi dal file e li visualizzano sul display.

Nel programma in BASIC 4.0 il file sequenziale è denominato SEQ. MAILDATA. Esso viene aperto in scrittura alla riga 40 e poi viene chiuso alla riga 220. Successivamente è riaperto per la lettura alla riga 310 e chiuso definitivamente alla riga 460.

Nella versione in BASIC < 3.0 il file è denominato MAILDATA < 3. È aperto in

scrittura alla riga 44 ed è chiuso alla riga 220; nella seconda parte del programma viene aperto in lettura alla riga 310 ed è chiuso alla riga 460.

Ambedue i programmi cancellano il file alla riga 470 così da permettere il loro riutilizzo. Ovviamente se questi programmi dovessero avere un uso concreto i file non dovrebbero essere cancellati, ma bensì ben conservati. In tal caso nomi e indirizzi nuovi sarebbero aggiunti al file mediante la procedura che vi descriveremo più avanti.

Per quanto concerne il controllo dello stato del dischetto troverete le istruzioni necessarie alle righe 50, 330 e 400 del programma in BASIC 4.0 e alle righe 41 e 42, 50 e 51, 320 e 321, 400 e 401 del programma in BASIC < 3.0.

Facciamo notare che il programma SEQ. MAIL. B < 3 apre il canale di comando alla riga 40, ma usa per la sua chiusura l'istruzione finale di STOP anche se sarebbe preferibile porre espressamente una CLOSE.

I programmi nelle due versioni di BASIC usano le stesse istruzioni per rilevere i dati dalla tastiera, scrivere il file sequenziale, leggere poi i dati dal file e visualizzarli sul display.

Le istruzioni dalla riga 60 alla 140 caricano i nomi e gli indirizzi dalla tastiera. In particolare si vede che i nomi e gli indirizzi entrano, come cinque campi separati, mediante il ciclo FOR NEXT dalla riga 70 alla 100. Mediante la stringa F$ si danno le indicazioni per l'ingresso dei dati all'operatore; i valori per F$ sono contenuti nell'istruzione DATA della riga 30 e sono ripristinati ad ogni ciclo mediante l'istruzione RESTORE della riga 110. I cinque campi per contenere ogni nome e indirizzo sono denominati AD$ (I).

Mediante le istruzioni dalla 120 alla 140 è possibile confermare i dati appena caricati oppure modificarli; questo tipo di "dialogo" lo abbiamo spesse volte descritto nel capitolo 5. Vi preghiamo notare però che la tecnica qui suggerita è molto semplice perchè a noi ora interessa solo curare la gestione dei file.

I nomi e gli indirizzi sono scritti nel file sequenziale con il ciclo dalla riga 160 alla 180. Ogni stringa viene scritta separatamente con la PRINT # della riga 170 ed è seguita da un carattere di ritorno del carrello. Questo ciclo di scrittura potrebbe essere sostituito con le due istruzioni seguenti:

```
160 C$=CHR$(13)
170 PRINT#1,AD$(1),C$,AD$(2),C$,AD$(3),C$,AD$(4),C$,AD$(5)
```

In tal caso l'istruzione INPUT # per la lettura successiva del file potrebbe essere modificata così:

```
390 INPUT#1,AD$(1),AD$(2),AD$(3),AD$(4),AD$(5)
```

Le istruzioni dalla riga 190 alla 210 permettono di caricare un altro nome e indirizzo oppure di passare alla seconda parte del programma.

Il ciclo dalla riga 370 alla 420 legge i cinque campi, di ogni nome e indirizzo, dal file sequenziale e poi li visualizza. Anche in questa fase vengono usati i valori contenuti nella DATA della riga 30 dalla READ della riga 380. La lettura del file avviene con la INPUT # della riga 390 e la stampa con la PRINT della riga 410.

Le istruzioni alla righe da 430 a 450 permettono all'operatore di visualizzare il successivo nome e indirizzo o terminare l'esecuzione del programma.

Vi facciamo notare infine che non abbiamo posta alcuna protezione contro l'eventuale richiesta di lettura di un altro nome e indirizzo una volta che il file sia stato chiuso. Questo inconveniente potrebbe essere risolto aggiungendo una nuova istruzione:

```
405 IF DS=64 THEN PRINT "FINE DEL FILE": I=5: GOTO 420
```

File sequenziali di dati misti

Per scrivere in un file sequenziale sia dati numerici che stringhe non è necessaria alcuna programmazione speciale. In ogni caso bisogna stare molto attenti a non confondere i due tipi di variabili; cioè leggere una variabile numerica con il nome di una stringa o viceversa.

Ecco un esempio di programma per scrivere due variabili numeriche e tre stringhe in un file sequenziale:

```
10 DOPEN#1,"DATA",W
20 C$=CHR$(13)
30 PRINT#1,P$,C$,X,C$,Q$,C$,Y,C$,R$
```

Queste cinque variabili potrebbero poi essere lette da una istruzione come la seguente:

```
100 INPUT#1,A$(1),X(1),A$(2),X(2),A$(3)
```

Se invece cambiassimo l'ordine delle variabili di stringa e di quelle numeriche, come in questo esempio, non avremmo più alcuna corretta assegnazione di valore alle variabili:

```
100 INPUT#1,A$(1),A$(2),A$(3),X(1),X(2)
```

AGGIUNTA DI DATI AD UN FILE SEQUENZIALE

Il BASIC 4.0 permette di aggiungere dati ad un file già esistente mediante le istruzioni APPEND # o CONCAT. L'istruzione APPEND aggiunge campi in coda ad un file; l'istruzione CONCAT concatena invece due file.

Istruzione APPEND per file sequenziali (BASIC 4.0)

Per illustrare l'istruzione APPEND # modifichiamo il programma SEQ NUM.B4 in SEQ.NUMAPPEND:

```
10 REM  PROGRAMMA "SEQ.NUMAPPEND"
15 REM
20 DOPEN#1,"TESTDATA",W
30 IF DS<>0 THEN PRINT DS$: STOP
35 FOR J=1 TO 3
40 REM SCRIVE DIECI RECORD
50 FOR I=1 TO 10
60 REM SCRIVE DIECI CAMPI PER RECORD
70 FOR C=1 TO 10
80 PRINT#1,(R-1)*100+C*J
85 IF DS<>0 THEN PRINT DS$: STOP
90 NEXT C
100 NEXT R
110 DCLOSE#1
200 REM LEGGE E VISUALIZZA IL CONTENUTO DEL FILE
210 DOPEN#1,"TESTDATA"
215 IF DS<>0 THEN PRINT DS$: STOP
220 FOR R=1 TO 10*J
230 PRINT "RECORD";R;
240 REM LEGGE E VISUALIZZA UN RECORD
250 FOR C=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$: STOP
270 PRINT N;
280 NEXT C
290 PRINT
300 NEXT R
310 DCLOSE#1
315 APPEND#1,"TESTDATA"
316 NEXT J
320 SCRATCH D0,"TESTDATA"
330 STOP
```

SEQ.NUMAPPEND è equivalente a tre esecuzioni di SEQ.NUM.B4. Durante ogni esecuzione vengono scritti 100 campi numerici nel file TESTDATA per cui alla fine il file contiene 300 campi.

Le tre esecuzioni sono effettuate dal ciclo FOR NEXT con indice J posto tra le righe 35 e 316.

Il contatore di campo C viene moltiplicato per J alla riga 80 per identificare i numeri aggiunti. Alla riga 220 l'estremo superiore del contatore di record R diviene 10★J poichè il numero totale di record aumenta di 10 ad ogni esecuzione.

È importante chiarire ora che l'istruzione APPEND # può operare solo su un file che sia stato già creato. Di conseguenza non è possibile sostituire semplicemente una istruzione APPEND # alla DOPEN # della riga 20 e ripetere tre volte l'apertura del file TESTDATA. È quindi necessario che inizialmente il file sia creato dalla DOPEN # e vengano scritti i primi dieci record. Successivamente il file sarà riaperto dalla APPEND # (riga 315) e verranno scritti i secondi dieci record; infine APPEND # apre ancora il file e vengono scritti gli ultimi record.

Se ora ponete in esecuzione il programma vedrete che inizialmente esso visualizzerà dieci record, come faceva il programma SEQ.NUM.B4, poi ne visualizzerà venti e da ultimo trenta. Vi sarà anche possibile riconoscere facilmente i nuovi record perchè contengono numeri con l'ultima cifra raddoppiata o triplicata in corrispondenza della seconda decina di record o della terza decina.

Concatenamento di file sequenziali (BASIC 4.0)

Il BASIC 4.0 con il DOS 2.0 permette di concatenare più file mediante l'istruzione CONCAT. Per darvi un'idea di come opera questa istruzione vi riportiamo questo programma dimostrativo CONCATEST:

```
5 REM  PROGRAMMA "CONCATEST"
6 REM
7 REM DIMOSTRAZIONE DELL'ISTRUZIONE CONCAT
8 REM
10 DOPEN#1,"DATA1",W
20 DOPEN#2,"DATA2",W
30 FOR I=1 TO 20
40 PRINT#1,I
50 PRINT#2,I+10
60 NEXT I
80 DCLOSE
90 DOPEN#1,"DATA1"
100 DOPEN#2,"DATA2"
110 PRINT "□"
120 FOR I=1 TO 20
130 INPUT#1,X: PRINTX;
140 NEXT
145 PRINT
150 FOR I=1 TO 20
160 INPUT#2,X: PRINTX;
170 NEXT
175 PRINT
180 DCLOSE
190 CONCAT "DATA2" TO "DATA1"
200 DOPEN#1,"DATA1"
210 FOR I=1 TO 40
220 INPUT#1,X: PRINT X;
230 NEXT
235 PRINT
240 DCLOSE
250 STOP
```

CONCATEST scrive venti numeri in due file sequenziali DATA1 e DATA2; poi concatena DATA2 a DATA1 e da ultimo visualizza sia i due file originali separati, sia DATA1 dopo la concatenazione.

I file DATA1 e DATA2 sono aperti alle righe 10 e 20. Il ciclo dalla riga 30 alla 60 scrive venti numeri in ognuno dei due file: i numeri dall'uno al venti nel primo e dall'unidici al trenta nel secondo.

I due file sono poi chiusi alla riga 80 con DCLOSE per essere riaperti con DOPEN # alle righe 90 e 100. Due cicli di stampa visualizzano il contenuto dei file: dalla riga 120 alla 140 e dalla 150 alla 170. Le PRINT delle righe 145 e 175 forzano un ritorno del carrello.

DATA1 e DATA2 sono chiusi alla riga 180 poi il secondo è concatenato al primo alla riga 190 con l'istruzione CONCAT. DATA1 viene infine aperto per essere letto e visualizzato. Alla riga 240 DATA1 è chiuso.

Tenete presente che CONCATEST non cancella i due file di dati al termine del programma. Se desiderate rieseguirlo dovete quindi cancellarli con SCRATCH, o in modo immediato, o nel programma con questa istruzione:

```
245 SCRATCH "DATA1":SCRATCH "DATA2"
```

Per quanto riguarda ancora l'istruzione CONCAT è opportuno che la usiate con molta attenzione perchè è molto facile commettere errori.

I due file da concatenare non devono essere vuoti e devono essere chiusi al momento del concatenamento. Se per errore cercate di concatenare un file vuoto il calcolatore entrerà in uno stato illimitato di attesa da cui potrà uscirne solo spegnendolo.

Se tentate di concatenare file ancora aperti o erroneamente chiusi, il calcolatore potrebbe anche aggiungere un file alla directory! Se questo si verifica vedrete il dischetto continuare la sua attività anche molto tempo dopo l'esecuzione della CONCAT. Potete fermare il dischetto premendo il tasto STOP sulla tastiera. Se ora provate a visualizzare la directory trovereste dell'informazione spuria dopo i nomi validi. Per ripristinare la directory corretta eseguite allora una istruzione COLLECT in modo immediato.

Istruzione APPEND per file sequenziali (BASIC < 3.0)

In BASIC < 3.0 per aggiungere dei nuovi dati ad un file sequenziale già esistente è necessario operare con due file. Questi due file li denominiamo DATA1 e DATA2 per praticità. Supponiamo allora di dover aggungere dati a DATA1. Creiamo allora un file temporaneo DATA2 e procediamo come segue:

1. Se DATA2 esiste già cancellatelo.
2. Aprite in scrittura DATA2.
3. Aprite DATA1 in lettura.
4. Leggete sequenzialmente i record di DATA1 e scriveteli su DATA2.
5. Appena incontrate il termine (EOF) di DATA1 iniziate a scrivere su DATA2 i nuovi record.
6. Chiudete DATA1 e poi cancellatelo.
7. Cambiate il nome di DATA2 in quello di DATA1.

Se desiderate aggiungere altri dati procedete ancora come sopra.

FINE DEL FILE (EOF)

Per controllare se siete giunti alla fine di un file dovete leggere se la parola di stato ST assume il valore 64. Per esempio potete porre la seguente istruzione:

```
200 IF ST=64 THEN PRINT "END OF FILE":STOP
```

FILE DI DATI RELATIVI (BASIC 4.0)

Solo il BASIC 4.0 può gestire i file di dati relativi. Su un file relativo aperto potete compiere sia operazioni di lettura che di scrittura. Tuttavia non potete leggere un file relativo vuoto, ma dovete prima avervi scritto qualcosa.

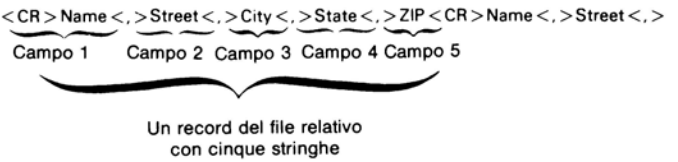
SEPARATORI DI CAMPO IN UN FILE RELATIVO

I caratteri virgola e ritorno del carrello hanno significati diversi, come separatori di campo, in un file relativo. **La lunghezza del record, indicata nell'istruzione DOPEN #** di un file relativo, identifica il numero di carattere in byte posti tra due ritorni del carrello contigui. Se tutti i campi sono separati da caratteri di ritorni del carrello, allora la lunghezza dei record diviene eguale a quella dei campi. Ricordatevi però che l'istruzione PRINT # nel BASIC 4.0 non trasmette automaticamente il ritorno del carrello, al termine della linea, se il numero del file è minore o eguale a 127.

Lunghezza dei record nei file relativi

Tutti i campi numerici sono generalmente terminati con il carattere di ritorno del carrello. Di conseguenza, se un file relativo contiene solo dati numerici, la lunghezza del record è anche la lunghezza del campo. Cioè il numero che viene posto dopo il parametro L nella DOPEN # identifica il numero di caratteri in byte che costituiscono ogni campo numerico.

Le variabili di stringa possono invece essere terminate con una virgola o un ritorno del carrello. Potete porre più stringhe in un unico record. Un indirizzo può avere per esempio cinque campi:



La lunghezza del record, che si deve indicare nell'istruzione DOPEN #, comprende in questo caso tutti e cinque i campi dell'indirizzo. Questo può essere molto utile perchè può soddisfare eventuali record con campo molto lunghi. Per esempio:

	Numero di caratteri					
	Nome Campo 1	Strada Campo 2	Città Campo 3	Stato Campo	CAP Campo 5	Totale
Indirizzo 1	9	14	16	2	5	46
Indirizzo 2	13	12	8	2	5	40
Indirizzo 3	12	11	12	2	5	42
Indirizzo 4	17	8	11	2	5	43
Indirizzo 5	10	12	13	2	5	42
etc.						

Se ogni indirizzo viene memorizzato in un singolo record è sufficiente scegliere come lunghezza di tutti i record 50.

Se invece ogni stringa viene terminata con un ritorno del carrello, allora campi e record avranno la stessa lunghezza. Tale lunghezza dovrà essere scelta così da soddisfare la stringa più lunga. Probabilmente nel nostro esempio dovremo scegliere una lunghezza dei campi di 20 byte. Per un indirizzo completo occuperemo quindi 100 byte e ciò significa che impegnamo una memoria doppia di quando invece separavamo i campi con le virgole.

Lettura dei record di un file relativo

Le istruzioni per leggere i campi di un file relativo sono INPUT # e GET #.

Se le stringhe usano la virgola come separatore e se l'istruzione di lettura è INPUT #, allora con tale istruzione leggerete tutte le stringhe comprese tra due caratteri di ritorno del carrello. Nelle pagine seguenti vi mostreremo alcuni esempi.

Se un file relativo contiene sia variabili di stringa che numeriche, allora definire la lunghezza del record diventa più difficile. Potete stabilire una lunghezza che permetta a più stringhe di essere scritte, una per campo, con le virgole come separatori, ma in tal caso ogni variabile numerica dovrebbe occupare un intero record. E quindi vi sarebbe un gran spreco di superficie del dischetto. Per risolvere questo problema vi sono due possibilità:

1. Scegliete una lunghezza del record adatta a contenere le variabili numeriche. Probabilmente questa lunghezza non sarà sufficiente per contenere le stringhe più lunghe per cui dovrete frazionarle.
2. Convertite le variabili numeriche in stringhe mediante la funzione STR$ e quindi usate la tecnica di registrazione delle stringhe già vista più sopra (questa seconda è senz'altro la tecnica più consigliabile).

SCRITTURA DI DATI NUMERICI IN UN FILE RELATIVO

Per studiare i file numerici relativi modifichiamo il programma SEQ.NUM.B4 creando il nuovo programma REL.NUM.B4:

```
10 REM PROGRAMMA "REL.NUM.B4"
20 DOPEN#1,"RELDATA",L10
30 IF DS<>0 THEN PRINT DS$: STOP
40 REM SCRIVE DIECI RECORD
50 FOR R=1 TO 10
60 REM SCRIVE DIECI CAMPI PER RECORD
70 FOR C=1 TO 10
80 PRINT#1,(R-1)*100+C
85 IF DS<>0 THEN PRINT DS$: STOP
90 NEXT C
100 NEXT R
110 DCLOSE#1
200 REM LEGGE IL FILE E LO VISUALIZZA
210 DOPEN#1,"RELDATA",L10
215 IF DS<>0 THEN PRINT DS$: STOP
220 FOR R=1 TO 10
```

```
230 PRINT "RECORD";R;
240 REM LEGGE E VISUALIZ. RECORD SUCCESSIVO
250 FOR C=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$: STOP
270 PRINT N;
280 NEXT C
290 PRINT
300 NEXT R
310 DCLOSE#1
320 SCRATCH D0,"RELDATA"
330 STOP
```

Per scrivere il nuovo programma vi consigliamo di caricare in memoria SEQ-.NUM.B4 e quindi di cambiare le linee 10, 20 e 210. Eseguitelo e se tutto è corretto otterrete lo stesso display del programma precedente per file sequenziali. Da ultimo salvate REL.NUM.B4.

Lunghezza del record

Notate che la lunghezza dei record, indicata nelle DOPEN del programma REL.NUM.B4, è abbastanza corta: solo 10 caratteri. Questo è dovuto al fatto che, dovendo scrivere valori numerici, ogni record equivale ad un campo e 10 caratteri sono sufficienti per la sua lunghezza.

Non sarebbe strettamente necessario chiudere il file RELDATA alla linea 110 e quindi riaprirlo alla linea 210. Noi lo abbiamo fatto per separare il programma in due moduli ed esaminare come le due parti interagiscono.

Cambiate ora la lunghezza del record, nell'istruzione DOPEN # della linea 210, dal valore L10 a L8 e rieseguite il programma. Il programma non funzionerà e apparirà il seguente messaggio:

```
50,RECORD NOT PRESENT, 00,00
BREAK IN 215
READY
```

Come potete immaginare l'errore è dovuto alla non corretta lunghezza del record nell'istruzione DOPEN # della linea 210. **In BASIC 4.0 non si possono riaprire file relativi con una lunghezza dei record errata (cioè diversa da quella dichiarata in precedenza).**

SCRITTURA DI STRINGHE IN UN FILE RELATIVO

Quando scrivete variabili di stringa in un file relativo potete terminare le singole stringhe sia con una virgola che con un ritorno del carrello. Se terminate ogni campo con un ritorno del carrello avrete una unica variabile di stringa per record. Se invece separate le stringhe con le virgole, allora potete porre quante stringhe volete in ogni record. In tal caso però dopo l'ultima stringa dovete porre ovviamente un ritorno del carrello.

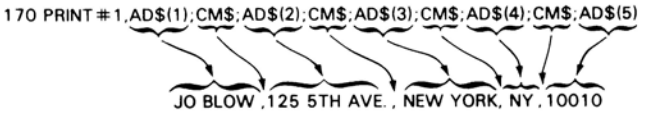
Come primo esempio di programma contenente un file relativo di stringhe, modifichiamo il programma SEQ.MAIL.B4 che si riferiva ad un file sequenziale. Nel nuovo programma il record sarà composto da cinque campi in cui saranno inseriti i nomi e gli indirizzi dell'elenco indirizzi che era gestito da SEQ.MAIL.B4. Il nuovo programma viene denominato REL.MAIL.B4 e qui di seguito ne diamo il listato:

```
10 REM PROGRAMMA "REL.MAIL.B4"
15 REM
20 REM QUESTO PROGRAMMA ILLUSTRA L'USO DI
21 REM FILE RELATIVI DI STRINGHE SU DISCHETTO
25 REM
30 DATA "   NOME: ","    VIA: "," CITTA': ","  STATO: ","    CAP:
40 DOPEN#1,"REL.MAILDATA",L50
50 IF DS<>0 THEN PRINT DS$: STOP
60 PRINT"□ BATTERE NOME E INDIRIZZO:□□"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT"BATTERE Y PER REGISTRARE; N PER RIPETERE I DATI";
130 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM SCRIVE NOME E INDIR. NEL FILE
160 CM$=CHR$(44)
170 PRINT#1,AD$(1);CM$;AD$(2);CM$;AD$(3);CM$;AD$(4);CM$;AD$(5)
171 IF DS<>0 THEN PRINT DS$: STOP
190 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
200 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="Y" THEN 60
220 DCLOSE#1
224 IF DS<>0 THEN PRINT DS$: STOP
300 REM VISUALIZZ. NOMI E INDIRIZZI
310 DOPEN#1,"REL.MAILDATA",L50
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM PULISCE LO SCHERMO E VISUALIZZA I DATI
350 PRINT"□□□"
360 RESTORE
365 INPUT#1,AD$(1),AD$(2),AD$(3),AD$(4),AD$(5)
366 IF DS<>0 THEN PRINT DS$: STOP
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
410 PRINT AD$(I)
420 NEXT I
430 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
440 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 440
450 IF Y$="Y" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"REL.MAILDATA"
480 STOP
```

Caricate il programma SEQ.MAIL.B4 in memoria dal dischetto, scrivete le nuove istruzioni e quindi eseguite il programma. Se tutto ciò è stato fatto correttamente esso si comporterà esattamente come SEQ.MAIL.B4. Se quindi REL.MAIL.B4 è privo di errori salvatelo su dischetto.

Esaminiamo ora le nuove istruzioni del programma REL.MAIL.B4. Le due istruzioni DOPEN # alle linee 40 e 310 sono state cambiate per indicare un file relativo con record di lunghezza 50 caratteri e con il nuovo nome REL.MAIL.B4.

I dati sono richiesti dalla tastiera e visualizzati come veniva fatto nel programma SEQ.MAIL.B4, ma le istruzioni che li scrivono sul file relativo sono completamente diverse. La PRINT # della linea 170 scrive un intero record. CM$ contiene il valore della virgola che le viene assegnato alla linea 160 tramite il codice ASCII (CHR$(44)). Notate che le variabili nella lista della PRINT # sono separate da punti e virgole. La combinazione di questi punti e virgole e delle CM$ poste tra ogni campo fanno sì che il record sia costituito nel modo seguente:



In questo esempio abbiamo posto che le variabili AD$ assumano dei valori concreti.

Notate che alla linea 171 abbiamo posto una istruzione di controllo dello stato del dischetto che sarà eseguita dopo ogni scrittura sullo stesso dischetto. Anche il programma SEQ.MAIL.B4 avrebbe potuto avere a questo punto una routine di controllo dello stato, ma ciò è invece di vitale importanza per i file relativi in quanto dovete sempre controllare che non avvengono superamenti ("overflow") dei record. Senza il controllo dello stato, alla linea 171, un eventuale nome/indirizzo che superi la lunghezza del record verrebbe registrato non correttamente e voi ve ne accorgereste solo rileggendo con molta attenzione il file. In realtà l'indicatore luminoso di errore del drive si accende in presenza di questi errori, ma per un tempo così breve che non potete accorgervene.

Per dimostrarvi quanto è importante l'istruzione della linea 171 provate a cancellarla e sostituite i punti e virgola, della linea 170, con virgole. Eseguite quindi il programma. L'indicatore di errore del drive lampeggerà ogni volta che viene scritto un record. Successivamente, quando i record saranno letti, vedrete che alcuni campi di ogni record sono andati persi.

Cerchiamo di capire che cosa è successo. Le virgole che avete posto nell'istruzione PRINT # hanno l'effetto di spostare la registrazione di una nuova variabile all'inizio della decima colonna successiva, come avviene per una normale PRINT. Siccome le variabili effettive della linea 170 sono nove (comprese le CM$), il record avrebbe dovuto essere lungo 90 caratteri almeno. Di conseguenza una parte del record è persa. Provate ad aggiungere la seguente istruzione per vedere l'effetto delle virgole:

```
PRINT AD$(1),CM$,AD$(2),CM$,AD$(3),CM$,AD$(4),CM$,AD$(5)
```

Ogni record sarà visualizzato esattamente come viene registrato. Se contate quindi

quanti caratteri è lungo ogni record vedrete dove avviene il troncamento. Ricordatevi infatti che la lunghezza del record è di 50 caratteri!

Alle linee 365 e 366 sono presenti le istruzioni per la lettura del file. Le linee 390 e 400 che leggevano invece il file sequenziale sono state tolte.

La lettura di un record dal dischetto avviene tramite l'istruzione INPUT # che legge sempre ciò che è compreso tra due caratteri di ritorno del carrello. Nel nostro programma REL.MAIL.B4 tra due caratteri di ritorno del carrello vi sono cinque campi per cui la INPUT # deve avere almeno cinque variabili nella sua lista.

Siccome nei nostri record sono registrate delle stringhe anche le variabili elencate nella INPUT # devono essere stringhe. Se così non fosse commettereste un errore di sintassi e il programma si fermerebbe.

Se nella INPUT # ponete meno di cinque variabili una parte del record non viene letta. Provate a togliere infatti AD$(4) e AD$(5) dalla linea 365 e vedrete che mancheranno completamente gli ultimi due campi dei record.

Analogamente provate ad aggiungere una nuova variabile AD$(6) alla INPUT#. Se in seguito rieseguite il programma potete constatare che nulla è cambiato. A differenza dei file sequenziali, in quelli relativi non viene assegnato alcun valore alle variabili della INPUT # poste in sovrannumero.

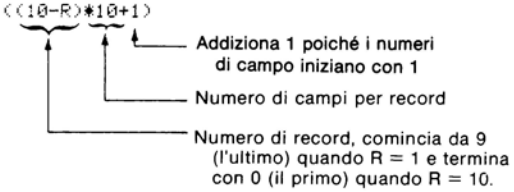
POSIZIONAMENTO NEI RECORD DI UN FILE RELATIVO

L'istruzione RECORD # vi permette di posizionarvi su ogni record e/o su ogni carattere (o byte) di un file relativo. Per dimostrarvi come funziona questa istruzione aggiungete la seguente istruzione al programma REL.NUM.B4:

```
23 RECORD#1,((10-R)*10+1)
```

Vedrete visualizzare dieci record con i numeri da 901 a 910 nel primo record e i numeri da 1 a 10 nell'ultimo. Questa visualizzazione è l'opposto di quella data da REL.NUM.B4.

Il fattore di posizionamento è ottenuto come segue:



L'istruzione RECORD # opera nello stesso modo sia che i record contengano dati numerici che stringhe. Per esercizio provate a inserire l'istruzione RECORD # nel programma REL.MAIL.B4 per selezionare nomi e indirizzi con qualunque sequenza.

Cambiamento di record in un file relativo

Se dovete cambiare un record, lo potete fare con la stessa PRINT # che avete usato per scriverlo. Posizionatevi sul record che vi interessa e scrivetevi sopra come se fosse la prima volta che lo create.

ISTRUZIONE GET # PER FILE SU DISCHETTO

L'istruzione GET # legge un solo carattere dal file su dischetto come la analoga GET legge un carattere dal buffer della tastiera. Anche in questo caso il carattere è prelevato dal buffer del dischetto a iniziare dalla prima delle sue 256 posizioni. Qualunque carattere viene trattato nello stesso modo cioè vengono letti anche gli spazi, i segni d'interpunzione, ecc.

Se usate l'istruzione GET # in un file sequenziale dovete leggere sequenzialmente a iniziare dal primo carattere. Se invece operate con un file relativo potete accedere direttamente a qualunque carattere di ogni record mediante l'istruzione RECORD# e iniziare a leggere dal carattere selezionato.

Evitate di usare GET # per leggere dati numerici da un file su disco. Ricordatevi infatti che GET # ritorna il valore 0 per un carattere numerico per cui non potete distinguere tra uno 0 vero e un carattere nullo.

Cerchiamo ora di illustrarvi l'uso dell'istruzione GET # modificando i programmi SEQ.MAIL.B4 e REL.MAIL.B4. Invece dell'istruzione INPUT # useremo GET # per leggere i nomi e gli indirizzi. Gli stessi cambiamenti si possono fare nel programma SEQ.MAIL.B < 3.

Uso di GET # nei file sequenziali

Per prima cosa sostituiamo la INPUT #, della linea 390 del programma SEQ.MAIL.B4, con GET #. L'istruzione GET # segue la stessa sintassi della GET che già conoscete. La nuova linea 390 sarà:

```
390 GET#1,AD$:IF AD$="" THEN 390
```

L'istruzione PRINT della linea 410 stamperà ora solo un carattere per cui dobbiamo aggiungere sia un punto e virgola alla sua fine, per evitare il ritorno del carrello, che una nuova istruzione per stabilire quando andare effettivamente a capo:

```
415 IF AD$<>CHR$(13) THEN 390
```

L'istruzione IF della linea 415 fa saltare indietro a GET # sino a che non viene trovato un ritorno del carrello. A questo punto il ciclo FOR NEXT può essere incrementato. Siccome il carattere di ritorno del carrello segna la fine di ogni parola, allora un ritorno del carrello è visualizzato dalla istruzione PRINT della linea 410 prima che l'IF della linea 415 proceda verso la parola successiva.

Caricate il programma SEQ.MAIL.B4 in memoria; fate i cambiamenti descritti ed eseguite il programma. L'esecuzione dovrebbe rimanere identica.

Per meglio comprendere il funzionamento di GET # provate a cambiare il programma così da modificare solo qualche carattere specifico. Per esempio fate apparire un carattere grafico sullo schermo ogni volta che si incontra un ritorno del carrello.

Uso di GET # nei file relativi

Qui di seguito trovate il programma REL. MAIL.GET # che deriva da REL.MAIL.B4. In esso trovate sia l'istruzione GET # che alcune modifiche per rendere trasparente l'organizzazione dei record relativi.

```
10 REM PROGRAMMA "REL.MAIL.GET#"
15 REM
20 REM QUESTO PROGRAMMA ILLUSTRA L'USO DI
21 REM FILE RELATIVI DI STRINGHE SU DISCHETTO
25 REM
30 DATA "   NOME: ","    VIA: "," CITTA': ","  STATO: " "    CAP: "
40 DOPEN#1,"REL.MAILDATA",L50
50 IF DS<>0 THEN PRINT DS$: STOP
60 PRINT"□ BATTERE NOME E INDIRIZZO:□□"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT"BATTERE Y PER REGISTRARE; N PER RIPETERE I DATI";
130 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM SCRIVE NOME E INDIR. NEL FILE
160 CM$=CHR$(44)
170 PRINT#1,AD$(1);CM$;AD$(2);CM$;AD$(3);CM$;AD$(4);CM$;AD$(5)
171 IF DS<>0 THEN PRINT DS$: STOP
190 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
200 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="Y" THEN 60
220 DCLOSE#1
224 IF DS<>0 THEN PRINT DS$: STOP
300 REM VISUALIZZ. NOMI E INDIRIZZI
310 DOPEN#1,"REL.MAILDATA",L50
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM PULISCE LO SCHERMO E VISUALIZZA I DATI
350 PRINT"□□□"
360 RESTORE
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390 GET#1,AD$: IF AD$="" THEN 390
395 IF DS<>0 THEN PRINT DS$: STOP
400 IF AD$=CHR$(32) THEN AD$="*"
405 IF AD$=CHR$(44) THEN PRINT AD$;: AD$=CHR$(13)
410 PRINT AD$(I);
415 IF AD$<>CHR$(13) THEN 390
420 NEXT I
430 PRINT"BATTERE Y PER ALTRO NOME E INDIR.; N PER TERMINARE";
440 GET Y$: IF Y$<>"Y" AND Y$<>"N" THEN 440
450 IF Y$="Y" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"REL.MAILDATA"
480 STOP
```

Dal momento che GET # legge un carattere alla volta, non dobbiamo preoccuparci molto della scelta dei separatori tra campi e tra record. Inoltre GET # può leggere i caratteri separatori come un qualunque altro carattere. Le istruzioni di INPUT # e controllo dello stato, delle linee 365 e 366 del programma REL. MAIL.B4, sono state tolte. Alle linee 390 e 395 compaiono le istruzioni GET # e quella di controllo dello stato.

Alla linea 400 gli spazi sono sostituiti da asterischi per renderli più evidenti.

Alla linea 405 si controlla la presenza delle virgole. Esse sono quindi visualizzate e poi sostituite con il ritorno del carrello.

Alla linea 410 è stato aggiunto un punto e virgola in coda alla PRINT poichè ora questa istruzione visualizza un carattere alla volta. Alla linea 415 il programma salta al successivo carattere sino a che non viene incontrato un ritorno del carrello. In tal caso il programma passa all'ingresso del prossimo campo. Ricordatevi che alla linea 405 le virgole sono state sostituite con ritorno del carrello per cui alla linea 415 sia le virgole che i ritorni del carrello causano un avanzamento al campo successivo.

Caricate ed eseguite il programma REL.MAIL.GET #. Noterete che esso crea la stessa visualizzazione sullo schermo di REL.MAIL.B4 salvo la presenza di asterischi invece degli spazi (non però dopo il codice CAP dell'indirizzo). Di conseguenza un carattere di ritorno del carrello apparirà direttamente dopo il CAP per cui avremo dello spazio inutilizzato sul dischetto tra un record e il successivo.

Uso delle istruzioni GET # e RECORD # nei file relativi

L'istruzione RECORD # permette di accedere ad ogni carattere di ogni record di un file relativo. Per constatare queste possibilità aggiungete la seguente linea al programma REL.MAIL.GET #:

```
365 RECORD#1,2,5
```

La seconda parte del programma REL.MAIL.GET # inizierà ora a visualizzare i nomi e gli indirizzi dal quinto carattere del secondo record.

FILE DI PROGRAMMI

I calcolatori CBM gestiscono i file di dati e quelli di programmi in modo completamente diverso e con differenti istruzioni.

Caricamento e memorizzazione dei file di programmi

I file di programmi sono caricati in memoria mediante l'istruzione LOAD per il BASIC < 3.0 e DLOAD per il BASIC 4.0.

I file di programmi sono memorizzati (o salvati) su dischetto mediante l'istruzione SAVE per il BASIC < 3.0 e DSAVE per il BASIC 4.0.

Caricamento e memorizzazione dei programmi sono argomenti già trattati nel capitolo 2.

Trattamento dei file di programmi come file di dati

Trattare i file di programmi, come se fossero file di dati, non è assolutamente vietato dalla sintassi dei BASIC CBM. Potete infatti eseguire le istruzioni di OPEN, CLOSE, GET #, INPUT # e PRINT # su un file di programma come se fosse un file di dati, ma non otterrete nessun buon risultato salvo che non siate dei programmatori molto esperti. In ogni caso ben difficilmente otterrete risultati migliori di quelli che già adesso potete avere con le normali istruzioni di gestione dei programmi o di editing.

Ricordatevi ancora che mediante l'istruzione OPEN in BASIC < 3.0 potete accedere ad un file di programma come se fosse un file di dati. Infatti se specificate l'indirizzo secondario 0 nella OPEN ciò equivale ad una operazione di LOAD del programma in memoria mentre se specificate l'indirizzo secondario 1 ottenete viceversa la memorizzazione (SAVE) del programma sul dischetto.

Duplicazione di file di programmi

Già altre volte vi abbiamo detto che è molto importante che abbiate una o più copie di ogni vostro programma. Quando vi è possibile cercate di tenere una seconda copia su un altro dischetto. Se le due copie fossero su uno stesso supporto potrebbero essere ambedue perse se tale supporto si rovinasse.

Per copiare un singolo file usate l'istruzione COPY in BASIC 4.0 oppure usate l'istruzione BACKUP per copiare l'intero dischetto.

In BASIC < 3.0 per copiare un file o l'intero dischetto dovete usare una variante dell'istruzione PRINT # come vi abbiamo descritto all'inizio di questo capitolo.

Sequenza di aggiornamento di un file di programma

Qualunque programma subisce delle migliorie continue per cui se ne hanno versioni sempre nuove. Noi vi consigliamo di mantenere sempre una copia della versione attuale e le due versioni precedenti (che chiamiamo "padre" e "nonno").

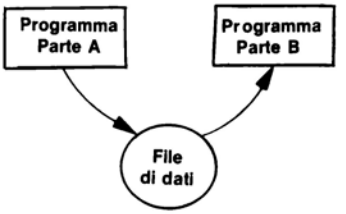
Quando aggiornate un programma seguite questa procedura:

1. Caricate (LOAD) la versione attuale in memoria e fate i cambiamenti voluti.
2. Cancellate (SCRATCH) il vecchio programma "nonno".
3. Denominate (RENAME) il programma "padre" come "nonno".
4. Denominate (RENAME) il programma attuale come "padre".
5. Salvate (SAVE) la nuova versione come programma corrente.

CODA DI LAVORI

Supponiamo che dobbiate scrivere un programma troppo lungo per la memoria che avete a disposizione. Vi mostriamo come potete risolvere il problema ripartendo il programma orginale, per esempio in due parti o moduli, che porrete in esecuzione separatamente. Per prima cosa scrivete le due parti in modo che siano completamente indipendenti salvo che la prima trasferisca i dati alla seconda tramite un file di dati esterno. Questo può essere illustrato così:

Supponiamo di chiamare le due parti di programma, o moduli, Parte A e Parte B; per eseguire l'intero programma dovete procedere come segue:



1. Caricate la Parte A in modo immediato mediante un comando LOAD.
2. Eseguite la Parte A con il comando RUN in modo immediato.
3. Prima della fine logica della Parte A il programma deve scrivere il file di dati che devono essere trasferiti alla Parte B.
4. Caricate la Parte B mediante un comando LOAD in modo immediato (in tale caso ricordiamo che viene anche automaticamente eseguita una NEW).
5. Eseguite la Parte B con un comando RUN in modo immediato.
6. La Parte B deve prevedere una lettura del file di dati (punto 3.) prima di iniziare la sua elaborazione.

Vi facciamo notare che con questa procedura la lunghezza dei due programmi A e B sono tra loro indipendenti, ma è necessario usare un file di transito per i dati ed intervenire manualmente per il caricamento e l'esecuzione della Parte B.

Dal momento che spesso è necessario ripartire in più di due parti un lavoro molto lungo e queste parti lavorano quasi sempre sugli stessi dati, risulta allora più opportuno operare con un metodo diverso che ora vi esponiamo e che rende automatico lo svolgimento successivo delle varie parti.

Ripartite il vostro programma in varie parti autonome e logicamente concluse. Ogni parte deve terminare logicamente con l'istruzione LOAD in BASIC < 3.0, o DLOAD in BASIC 4.0, della parte successiva. Ricordiamo che la LOAD o la DLOAD, eseguite da programma, caricano e pongono in esecuzione senza eseguire automaticamente una NEW. Se vi è possibile fate in modo che la prima parte sia uguale o più lunga di tutte le altre. Infatti, quando va in esecuzione la prima parte, vengono posizionati i puntatori alle aree di memoria delle variabili e se le parti

successive non superano la lunghezza della prima le variabili restano indisturbate. Se desiderate ottimizzare il funzionamento di tutto il lavoro, è consigliabile definire tutte le variabili nella prima parte pur non essendo ciò strettamente necessario.

Se non è possibile soddisfare la condizione che la prima parte sia la più lunga, è necessario operare sui puntatori, all'inizio della prima parte, con delle istruzioni POKE che spostino l'inizio della zona variabili in modo da lasciare la zona programma sufficientemente capiente per contenere la parte più lunga.

Potete osservare che questa seconda procedura richiede un intervento manuale solo all'inizio per caricare ed eseguire la prima parte.

PROGRAMMAZIONE DELLA STAMPANTE

Fino a questo punto abbiamo usato molto poco la stampante del nostro calcolatore, salvo quando abbiamo listato un programma. Ma questa non è l'applicazione principale di una stampante. Infatti quasi tutti i lavori di programmazione alla fine terminano con dei dati riportati su un tabulato. È importante quindi che questi tabulati si presentino nella forma più corretta possibile ed anche più facilmente leggibile.

Le stampanti che si possono collegare ad un calcolatore CBM sono molte, ma in questo testo prendiamo in esame solo il modello 2022 e il modello 2023. Ambedue sono dotate di microprocessore e questo è il motivo per cui sono facili da usare.

Le due stampanti modello 2022 e 2023 impiegano lo stesso insieme di caratteri della tastiera dei PET.

Per accedere alle stampanti dovete aprire un file logico specificando l'unità fisica #4 e un indirizzo secondario compreso tra 0 e 7. Se l'indirizzo secondario non è indicato, allora viene posto eguale a 0. Come riportato nella Tabella 6-4 l'indirizzo secondario permette di scegliere tra queste opzioni:

- Stampare i dati esattamente come sono ricevuti.
- Stampare i dati secondo un determinato formato.
- Stabilire il numero di linee di stampa per pagina.
- Specificare lo spazio di interlinea.
- Stampare caratteri che non sono compresi nell'insieme di caratteri standard.
- Abilitare la stampa di messaggi diagnostici speciali.

Nella Tabella 6-5 sono riportati i caratteri di controllo speciali che permettono ulteriori formati di stampa.

STAMPA DI DATI COSI' COME SONO RICEVUTI

Per stampare i dati così come sono ricevuti dovete aprire un file logico con l'unità fisica # 4 e indirizzo secondario 0 oppure non indicare nessun indirizzo secondario. Successivamente potete stampare i dati con PRINT # e/o CMD.

Stampa con l'istruzione PRINT #.

L'istruzione PRINT # invia dati alla stampante così come li manderebbe ad una cassetta o a un dischetto. Per esempio per stampare la parola "MESSAGGIO" provate a caricare il seguente programma:

```
10 OPEN 2,4
20 PRINT#2, "MESSAGGIO"
30 CLOSE 2
40 STOP
```

Ogni volta che lo eseguite vi appare la parola "MESSAGGIO" sulla stampante e questo avviso sullo schermo:

```
BREAK IN 40
READY
```

perchè vi è uno STOP alla linea 40.

Attenzione! Non potete usare le istruzioni DOPEN # e DCLOSE # del BASIC 4.0 per accedere ad una stampante; queste istruzioni lavorano solo con i dischetti.

Stampa con l'istruzione CMD

Invece di usare l'istruzione PRINT # potete usare l'istruzione CMD per inviare dati ad una stampante. CMD deve però essere seguita da almeno una PRINT # prima che il file logico sia chiuso. Per dimostrarvi l'uso di CMD caricate ed eseguite questo programma:

```
10 OPEN 2,4
20 CMD 2, "MESSAGGIO"
25 PRINT#2
30 CLOSE 2
40 STOP
```

Quando questo programma sarà eseguito vedrete stampare la parola "MESSAGGIO" seguita da due ritorni del carrello. La PRINT # della linea 25 genera infatti il secondo ritorno del carrello.

Stampa con le istruzioni CMD e PRINT

Dopo un comando CMD l'istruzione PRINT visualizza i suoi dati sulla stampante e non sul display e questo sin'tanto che non sia eseguita una istruzione PRINT #. Per farvi vedere come questo avviene eseguite questo programma:

```
10 OPEN 2,4
20 CMD 2
21 PRINT "MESSAGGIO"
25 PRINT#2
26 PRINT "MESSAGGIO"
30 CLOSE 2
40 STOP
```

Tabella 6 - 4: Caratteri di controllo delle stampanti, usati nei testi di stampa con indirizzo secondario 0 o 1.

	Funzione di stampa	Carattere codice	Tasto	Commenti
Necessarie	Fine del campo di stringa	CHR$ (29)	CURSORE →	Ogni campo di stringa deve essere seguito da CHR$ (29) eccetto per l'ultima variabile nella lista di PRINT #.
	Spazi bianchi anteriori ad una stringa	CHR$ (160)	SHIFT + SPAZIO	Per inserire spazi bianchi anteriormente ad una stringa dovete usare le spaziature in modo shiftato. Spaziature non shiftate sono trascurate.
Opzionali	Allargamento dei caratteri	CHR$ (1)	Nessuno	La presenza di un carattere CHR$ (1) nella lista di una PRINT # raddoppia la larghezza di tutti i caratteri seguenti stampanti sulla stessa riga (i caratteri normali hanno 6 punti di larghezza). Un ritorno del carrello annulla gli effetti di ha R$(1). Più CHR$(1) hanno effetti cumulativi.
	Avanzamento riga (Line Feed)	CHR$(10)	Nessuno	La stampante esegue un ritorno del carrello e l'avanzamento di una riga.
	Ritorno del carrello	CHR$(13)	RETURN	
	Caratteri invertiti	CHR$(18)	OFF RVS	Vengono stampati caratteri in campo invertito. Questo comando è annullato dal ritorno del carrello.
	Lettere minuscole	CHR$(17)	CURSORE↓	Vengono stampate lettere minuscole. Questo comando è annullato da un ritorno diel carrello o da un comando per lettere maiuscole.
	Inizio pagina	CHR$(19)	HOME	Porta la carta a inizio della pagina se è abilitata l'impaginatura.
	Virgolette	CHR$(34)	Nessuno	Permette la stampa di tutti i caratteri di controllo.
	Ripristino larghezza normale dei caratteri	CHR$(129)	Nessuno	Annulla il comando CHR$(1) di allargamento dei caratteri
	Ritorno del carrello senza avanzamento della riga	CHR$(141)	Nessuno	La stampante esegue un ritorno del carrello, ma non l'avanzamento di una riga. Questo permette di riscrivere su una stessa riga.
	Lettere maiuscole	CHR$(145)	CURSORE↑	Se in precedenza era stato dato un comando per lettere minuscole CHR$(17), ora vengono stampate quelle maiuscole. Non ha effetto se la stampante stava già usando le lettere maiuscole.
	Caratteri non invertiti	CHR$(146)	SHIFT + OFF RVS	Annulla il comando di caratteri invertiti.
	Impaginatura	CHR$(147)	CLR	Stampa 66 righe per pagina sino a quando una successiva uscita con indirizzo secondario 3 cambia il numero di righe per pagina.

Tabella 6-5: Caratteri di formato delle stampanti, usati nelle istruzioni di formato con indirizzo secondario 2.

Tipo di dati	Codice	Specifiche	Impiego	Esempi		
				Dato	Formato	Risultato di stampa
Campi numerici	9	Cifre numeriche senza zeri anteriori	Riporta tutte le cifre alla sinistra e alla destra del punto decimale (se presente).	23 124.756 124.756	999.9 9999.99 999	23.0 124.75
	Z	Cifre numeriche con zeri	Riporta tutte le cifre alla sinistra del punto decimale (se presente).	23 124.756 124.756	ZZZ.9 ZZZZ.99 ZZZ	023.0 0124.75 124
		Punto decimale	I numeri sono allineati rispetto al punto decimale (se presente). Altrimenti sono allineati a destra.		vedere sotto	
	$	L'importo è in $ americani	Stampa il carattere $ davanti alla prima cifra non nulla o nella prima posizione del campo.	23 124.756	$999.9 $$$$$.99	$23.0 $124.75
	S	Numero con segno. S deve essere il primo carattere del formato	Il segno (+ o –) viene stampato come primo carattere non nullo.	124.756 –23 –475.2	S$999.99 S$$$$.99 S9999.9	+$124.75 –$23.00 +475.2
	–	Numero con segno; "–" deve essere l'ultimo carattere del formato	I numeri negativi sono segnati con un "–" posto come ultimo carattere non nullo.	124.756 –23 –475.2	$999.99– $$$$.99– 9999.9–	$124.75 $23.00- 475.2–
Stringhe	A	Qualunque posizione di un campo di stringa.	"A" indica le posizioni dei caratteri di una stringa. Nel campo consentito le stringhe sono allineate a sinistra.	ABCDE ABC ABCDE	AAAAAA AAAAAA AA	ABCDE ABC AB
Qualunque	b	Spaziatura	Usare spazi vuoti b per separare i campi.			
	<RVS ON>	Stampa il prossimo carattere	Il carattere, specificato nel formato, viene stampato.		<RVS ON> – <RVS ON> –A	– –

Vedrete allora la stampante andare a capo, scrivere la parola "MESSAGGIO" e poi andare ancora a capo due volte. Successivamente vedrete apparire anche sullo schermo la parola "MESSAGGIO". Il comando CMD ha causato il primo ritorno del carrello. La PRINT della linea 21 fa scrivere alla stampante la parola "MESSAGGIO" con ritorno a capo. La PRINT # della linea 25 causa il secondo ritorno del carrello. La PRINT della linea 26 visualizza sul display ancora la parola "MESSAGGIO".

Provate ora a togliere la PRINT della linea 21. Vedrete che la stampante va due volte a capo, ma la parola "MESSAGGIO" invece di essere stampata appare solo sul display.

Confronto tra le istruzioni CMD e PRINT #.

Dobbiamo ora chiarire quale è la differenza tra le istruzioni CMD e PRINT #.

Pensate alla stampante come ad un sostituto del display. Un unico canale di uscita va dal calcolatore sia al display che alla stampante. Quando eseguite una istruzione OPEN con unità fisica 4, voi avvisate il calcolatore che è anche presente una stampante, ma il canale di uscita continua a selezionare il display.

Quando viene eseguita una PRINT # il canale di uscita viene momentaneamente commutato dal display alla stampante, per i valori in quel momento trasferiti, poi ritorna a collegarsi con il display.

Se invece della PRINT # fate eseguire una CMD il canale di uscita viene definitivamente commutato sulla stampante e il display non ha più un suo canale. Se infatti eseguite ora delle semplici PRINT, i dati non andranno sul display ma sulla stampante.

Questa situazione rimane sin quando non venga eseguita una nuova PRINT # che ristabilisce il collegamento con il display.

Il collegamento con una stampante deve alla fine essere chiuso come ogni altro file. E questo equivale a dire al calcolatore che la stampante non è più presente.

Se erroneamente il canale di uscita è rimasto commutato verso la stampante dopo la chiusura del file, allora le PRINT successive continueranno ad andare sulla stampante. Provate infatti ad eseguire questo programma:

```
10 OPEN 2,4
20 CMD 2
30 CLOSE 2
35 PRINT "MESSAGGIO"
40 STOP
```

In esecuzione otterrete il seguente tabulato sulla stampante:

```
MESSAGGIO
BREAK IN 40
READY
```

Cioè i messaggi che dovevano andare sul display sono andati invece sulla stampante.

STAMPA DI DATI CON FORMATO

Per prima cosa è necessario che stabiliate un formato. **Mediante le indicazioni della Tabella 6-5 definite una stringa di testo, che rappresenti il formato da voi scelto, e poi la trasmettete alla stampante mediante l'indirizzo secondario 2.**

I dati da stampare devono invece essere trasferiti mediante l'indirizzo secondario 1. Se non avete specificato il formato allora i dati saranno stampati come se aveste posto l'indirizzo secondario 0.

Per programmare quindi una stampante con formato dovete aprire due file logici: il primo con unità fisica 4 e indirizzo secondario 2, il secondo sempre con unità fisica 4, ma con indirizzo secondario 1. Tramite questi due canali trasferite poi la stringa che rappresenta il formato e i dati.

Stampa di dati numerici con formato

Cominciamo ad esaminare la stampa di valori numerici.

Per dare le posizioni delle singole cifre del nostro numero sono stati scelti i caratteri 9, Z e punto decimale.

Il punto decimale, se viene indicato nel formato, sarà stampato nella stessa posizione nel campo numerico.

I caratteri 9 e Z specificano ambedue le posizioni delle cifre. Tuttavia la Z forzerà la stampa degli zero, mentre invece il 9 lascerà degli spazi bianchi. Ecco per maggior chiarezza alcuni esempi:

Numero	Formato	Risultato
123.456 6457 -128.1	999999.99	123.45 6457.00 128.10
123.456 6457 -128.1	ZZZZZ.9	00123.4 00123.4 00128.1

Un numero può essere stampato con il segno posto anteriormente o in coda.

Una lettera S all'inizio del formato numerico farà comparire il segno + o − posto anteriormente al numero.

Un segno − posto come ultimo carattere del formato farà apparire un segno meno in coda al numero se negativo; non è possibile fare apparire analogamente il segno + in coda ad un numero positivo.

Quando un numero è rappresentativo di valori monetari in dollari $, allora il segno $ può precedere il numero oppure può essere posto all'inizio dell'intero campo numerico previsto. Se è previsto il segno + o −, allora esso può precedere il segno di dollaro oppure essere posto in coda al numero.

Per fare alcuni semplici esempi provate a porre il segno $ all'inizio del formato numerico. In questo caso avrete la stampa di $ nella posizione più a sinistra del

campo numerico. Se dovete prevedere anche il segno aritmetico, allora il formato inizierà con S$ e in stampa avrete prima il segno e poi $.

Se volete che il segno di dollaro sia attaccato alla prima cifra ponete nel formato il segno $ in tutte le posizioni che precedono il punto decimale ed uno in più per tenere conto del segno $. Anche in questo caso potete iniziare il formato con una S per indicare il segno aritmetico.

Ecco in conclusione alcuni esempi:

Numero	Formato	Risultato
123.456 6457 -128.1	S9999	123 6457 -128
123.456 6457 -128.1	S$9999.99	$0123.45 $6457.00 -$0128.10
123.456 6457 -128.1	S$9999.99	$123.45 $6457.00 -128.10
123.456 6457 -128.1	$$$$$.99-	$123.45 $6457.00 $ 128.10-
123.456 6457 -128.1	$$$$$.99-	$0123.45 $6457.00 $0128.10-

Più oltre vi indicheremo come sostituire il segno monetario di dollaro con altri segni a seconda delle vostre necessità.

Per mostrarvi come può essere fatto un programma di stampa con formati numerici vi riportiamo qui di seguito il programa NUM.FORM.PRINT che legge otto valori numerici contenuti nella DATA della linea 30 e poi li stampa con il formato specificato dalla PRINT # dalla linea 100. Quando ponete in esecuzione questo programma vedrete che esso fa stampare gli otto numeri in colonna verticale.

```
10 REM PROGRAMMA "NUM.FORM.PRINT"
20 REM ESEMPIO DI STAMPA NUMERICA CON FORMATO
30 DATA 1.75,-12300,0.74682,12,-456.832,23456.78,-100.798,4789326
70 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
80 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
90 REM USCITA DEL FORMATO
100 PRINT#2,"999999.99"
110 FOR I=1 TO 8
120 READ N
130 PRINT#1,N
140 NEXT I
150 CLOSE 1
155 CLOSE 2
160 STOP
```

```
     1.75
 12300.00
      .74
    12.00
   456.83
 23456.78
   100.79
******.**
```

Notate che i numeri sono stati allineati sul punto decimale. **Se un numero non può essere rappresentato con il formato prefissato, allora il calcolatore pone semplicemente degli asterischi.** Questo è avvenuto per l'ottavo numero dell'esempio.

Provate ora a cambiare il formato e porre delle Z al posto dei 9 che precedono il punto decimale. Otterrete questo tabulato:

```
000001.75
012300.00
000000.74
000012.00
000456.83
023456.78
000100.79
******.**
```

Notate che la Z fa stampare gli zeri nei posti vuoti. L'ottavo numero sarà ancora sostituito da asterischi perchè non ha un formato sufficiente. Provate allora ad aggiungere una posizione nel formato a sinistra del punto decimale e vedrete che anche l'ultimo numero sarà stampato.

Non potete mescolare la Z e i 9 a sinistra del punto decimale. Se lo fate il calcolatore interpreterà il formato a iniziare dalla sinistra fino a dove vede un cambiamento. Con un esempio cerchiamo di essere più chiari:

```
100 PRINT#2,"ZZZ999.99"
```

Se ora utilizzate questo nuovo formato vedrete che il calcolatore lo interpreta come se fosse "ZZZZ". Se invece avete posto:

```
100 PRINT#2,"9999ZZZ.99"
```

per il calcolatore significa "9999".

Se volete evidenziare il segno potete porre una S all'inizio della stringa di formato:

```
100 PRINT#2,"S9999999.99"
```

Ponete in esecuzione il programma con questa nuova linea e ottenete:

```
+      1.75
-  12300.00
+       .74
+     12.00
-    456.83
+  23456.78
-    100.79
+4789326.00
```

Se invece il segno deve comparire in coda al numero, ponete un meno in coda alla stringa del formato (ricordatevi però che in questo caso solo il segno meno viene evidenziato):

```
100 PRINT#2,"9999999.99-"
```

Eseguite, con quest'ultima linea 100, il programma e vedrete che i valori negativi portano in coda il segno meno.

È importante far notare che il calcolatore tronca la parte decimale di un numero e non la arrotonda. Cioè stampa tante cifre decimali quante sono quelle indicate nel formato.

Se i valori che dobbiamo stampare sono monetari poniamo allora il segno $ davanti al formato, eventualmente preceduto dalla S per indicare anche il segno. Nel nostro esempio possiamo scrivere:

```
100 PRINT#2,"S$9999999.99"
```

e in esecuzione otteniamo:

```
+$       1.75
-$   12300.00
+$        .74
+$      12.00
-$     456.83
+$   23456.78
-$     100.79
+$4789326.00
```

Se per errore avessimo posto la S dopo $, il calcolatore avrebbe scritto tutti i numeri senza formato.

Se per motivi commerciali il vostro tabulato deve avere gli importi negativi con il segno meno posto in coda ai numeri, allora invece della S ponete un segno meno in coda al formato:

```
100 PRINT#2,"$9999999.99-"
```

Il tabulato apparirà quindi così:

```
$       1.75
$   12300.00-
$        .74
$      12.00
$     456.83-
$   23456.78
$     100.79-
$4789326.00
```

Da ultimo vediamo la possibilità di far stampare il segno $ attaccato direttamente al numero. Ponete in tal caso il segno $ in tutte le posizioni a sinistra del punto decimale nel formato:

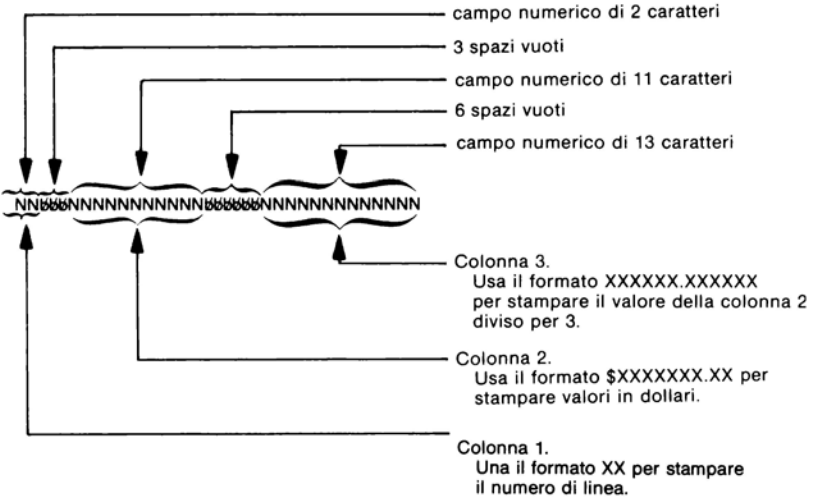
```
100 PRINT#2,"$$$$$$$.99-"
```

In esecuzione ottenete:

```
    $1.75
$12300.00-
    $.74
   $12.00
  $456.83-
$23456.78
  $100.79-
******.****
```

L'ottavo valore è andato perso poichè il formato era insufficiente. Infatti tale numero, che ha sette cifre nella parte intera, necessita di otto segni $: sette per le cifre e uno per il segno di dollaro.

Finora abbiamo descritto i possibili formati per avere un tabulato con i numeri su una unica colonna. Se desideriamo invece avere più colonne è sufficiente aggiungere i formati delle nuove colonne separati tra loro con spazi bianchi. Per chiarirvi come impostare un formato di questo tipo vi diamo qui di seguito lo schema grafico di un formato per tre colonne:



L'istruzione PRINT # del nostro esempio può essere riscritta così:

```
100 PRINT#2,"99   $$$$$$$$.99-    999999.999999"
```

Cambieremo anche la PRINT # della linea 130 per generare i numeri che porremo nelle tre colonne e cioè I, N e N/3:

```
130 PRINT#1,I,N,N/3
```

Quando eseguirete il programma otterrete questo tabulato:

```
1          $1.75            .583333
2     $12300.00-        4100.000000
3          $.74            .248940
4        $12.00           4.000000
5       $456.83-        152.277333
6     $23456.78        7818.926670
7       $100.79-         33.599333
8   $4789326.00       ******.******
```

Voi stessi potete constatare che il numero di spazi tra una colonna e l'altra è uguale a quello degli spazi indicati nella PRINT # del formato. In questo caso particolare vi sono anche tra le colonne gli spazi lasciati vuoti dai formati dei singoli numeri.

Stampa di stringhe con formato

Per definire la posizione dei caratteri di una stringa si usa la lettera A mentre per definire la spaziatura tra due stringhe si usa lo spazio bianco come per i formati numerici. Anche i formati per le variabili di stringa vengono trasmessi tramite una stringa posta nella lista di una PRINT # che faccia riferimento ad un file logico precedentemente aperto con il numero di unità fisica 4 e indirizzo secondario 2.

Le variabili di stringa, che devono essere stampate con il formato stabilito, saranno trasmesse da un'altra PRINT # il cui file logico è stato aperto con il numero 4 per l'unità fisica e 1 per l'indirizzo secondario. **Le stringhe in questa seconda PRINT # devono essere separate dal carattere CHR$(29) che può essere generato con il tasto CURSOR RIGHT nell'ambito di una stringa. Le stringhe saranno poi allineate a sinistra e gli eventuali spazi lasciati liberi a destra saranno riempiti da spazi vuoti. Spazi vuoti a sinistra delle stringhe saranno troncati.**

Ecco un esempio di formato:

```
100 PRINT#X,"AAAAAAAAAA     AAAAAAAAAAAA"
110 PRINT#Y,M$CHR$(29)N$
```

Dove X rappresenta un file logico aperto con numero di unità fisica 4 e indirizzo secondario 2 mentre Y rappresenta un file logico con numero di unità fisica 4 e indirizzo secondario 1.

L'istruzione PRINT # X specifica due campi di stringa di 10 e 12 caratteri separati da 5 spazi.

L'istruzione PRINT # Y specifica due stringhe M$ e N$ separate da CHR$(29) come richiesto dalla sintassi CBM. **Notate che in questa seconda PRINT # non abbiamo posto le virgole per separare gli elementi della sua lista.**

Se volete potete porre le virgole; la seguente istruzione è infatti valida:

```
PRINT#Y,M$,CHR$(29),N$
```

Le stringhe M$ e N$ possono essere sostituite con valori immediati ed inoltre

potete, oppure no, porre le virgole per separare le stringhe stesse e il separatore CHR$(29). Ecco un esempio:

```
PRINT#Y,"ONE"CHR$(29)"TWO"
```

Per illustrarvi concretamente un esempio di stampa di stringhe con formato modifichiamo il programma NUM.FORM.PRINT e generiamo il nuovo programma STR.FORM.PRINT:

```
10 REM PROGRAMMA "STR.FORM.PRINT"
20 REM ESEMPIO DI STAMPA DI STRINGHE CON FORMATO
30 DATA "MARY PERKINS","35 WEST ST.","BERKELEY","CALIFORNIA","94705"
35 DATA "345-67-8910","SPONSOR","AXC"
70 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
80 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
90 REM USCITA DEL FORMATO
100 PRINT#2,"AAAAAAAAAA      AAAAAAAAAAAA"
105 SP$=CHR$(29)
110 FOR I=1 TO 4
120 READ M$,N$
130 PRINT#1,M$,SP$,N$
140 NEXT I
150 CLOSE 1
155 CLOSE 2
160 STOP
```

L'istruzione PRINT # X appare alla linea 100 con file logico 2 aperto alla linea 80. PRINT # Y appare alla linea 130 con file logico 1 aperto alla linea 70. Invece di porre espressamente CHR$(29) nella PRINT # 1 della linea 130 abbiamo posto SP$ eguagliato a CHR$(29) nella linea 105.

Gli otto valori che dovranno essere tabulati sono definiti nelle due DATA alle linee 30 e 35. Essi rappresentano un nome con indirizzo seguito da un numero di matricola e da due parole in codice.

Notate che il primo campo del formato tronca il cognome dell'esempio perchè il formato è troppo corto. Aggiungete quindi delle A nel formato per ottenere una stampa completa del cognome.

Notate anche che le stringhe sono allineate a sinistra a differenza degli esempi numerici in cui erano allineati a destra.

Per stampare spazi vuoti innanzi alle stringhe non è possibile inserirli davanti alle stesse stringhe durante la stesura del programma. È possibile però risolvere questo problema usando il carattere CHR$(160) che rappresenta una spaziatura con shift abbassato. Vediamo le cose con un esempio. Dapprima provate a inserire due spazi davanti a AXC nella linea 35:

```
35 DATA "345-67-8910","SPONSOR","  AXC"
```

Premere due volte la barra di spazio

Se ora eseguite il programma non vedrete cambiare niente. I due spazi davanti a AXC sono stati troncati. Se invece ribattete la linea 35 con due spaziature "in maiuscolo" (cioè shift abbassato) e poi eseguite il programma vedrete finalmente i due spazi davanti a AXC.

Vediamo ora che mediante il concatenamento di stringhe è possibile sposta o allineare a destra le stringhe. Vi illustriamo questo mediante la seguente modifica del programma STR.FORM.PRINT:

```
10 REM PROGRAMMA "STR.FORM.PRINT"
20 REM ESEMPIO DI STAMPA DI STRINGHE CON FORMATO
30 DATA "MARY PERKINS","35 WEST ST.","BERKELEY","CALIFORNIA","94705"
35 DATA "345-67-8910","SPONSOR","AXC"
70 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
80 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
90 REM USCITA DEL FORMATO
100 PRINT#2,"AAAAAAAAAA     AAAAAAAAAAAA"
105 SP$=CHR$(29)
106 BL$="            ": REM 12 SPAZI CON SHIFT
110 FOR I=1 TO 4
120 READ M$,N$
125 IF LEN(M$)<10 THEN M$=LEFT$(BL$,(10-LEN(M$)))+M$
126 IF LEN(N$)<12 THEN N$=LEFT$(BL$,(12-LEN(N$)))+N$
130 PRINT#1,M$ SP$ N$
140 NEXT I
150 CLOSE 1
155 CLOSE 2
160 STOP
```

Per prima cosa alle linee 125 e 126 si individuano quelle stringhe di lunghezza inferiore a quella del campo previsto dal formato. Alle stringhe più corte si aggiungono a sinistra degli spazi "maiuscoli" tanti quanti necessari per portare alla lunghezza massima la stringa. Gli spazi "maiuscoli" sono prelevati dalla stringa BL$, definita alla linea 106, mediante la funzione LEFT$.

Vediamo adesso come modificare il programma STR.FORM.PRINT per ottenere un formato di stampa più elegante. Per esempio i cinque campi per il nome e l'indirizzo (senza caratteri troncati) possono essere posti in colonna mentre gli altri tre campi è preferibile porli su una unica linea. Denominato STR.FORM.PRINT1 il nuovo programma di cui diamo il listato e un esempio di esecuzione:

```
10 REM PROGRAMMA "STR.FORM.PRINT2"
20 REM ESEMPIO DI STAMPA DI STRINGHE CON FORMATO
30 DATA "MARY PERKINS","35 WEST ST.","BERKELEY","CALIFORNIA","94705"
35 DATA "345-67-8910","SPONSOR","AXC"
70 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
80 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
90 REM USCITA DEL FORMATO
105 SP$=CHR$(29)
110 FOR I=1 TO 8
120 READ M$
140 NEXT I
150 PRINT#2,"99  AAAAAAAAAAAAAA"
160 FOR I=1 TO 5
170 PRINT#1,M$(I)
180 NEXT I
190 PRINT#2,"99  AAAAAAAAAAA   AAAAAAA   AAA"
200 PRINT#1,M$(6)SP$M$(7)SP$M$(8)
210 CLOSE 1
220 CLOSE 2
230 STOP
```

```
MARY PERKINS
35 WEST ST.
BERKELEY
CALIFORNIA
94705
345-67-8910   SPONSOR   AXC
```

Le otto stringhe delle due DATA sono trasferite nel vettore M$(I) con il ciclo FOR NEXT dalla linea 110 alla 140. Cinque di questi valori sono allora stampati in colonna verticale, con il ciclo FOR NEXT dalla linea 160 alla 180 e formato definito alla linea 150. Un nuovo formato viene quindi definito alla linea 190 per essere poi usato nella stampa delle altre tre stringhe con la PRINT # della linea 200.

Stampa di dati numerici e di stringhe con formato

Potete mescolare dati numerici e stringhe in una stampa con formato. Per illustrarvi come questo sia possibile vi diamo qui di seguito un programma dimostrativo denominato STR.FORM.PRINT2:

```
10 REM PROGRAMMA "STR.FORM.PRINT1"
20 REM ESEMPIO DI STAMPA DI STRINGHE CON FORMATO
30 DATA "MARY PERKINS","35 WEST ST.","BERKELEY","CALIFORNIA","94705"
35 DATA "345-67-8910","SPONSOR","AXC"
70 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
80 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
90 REM USCITA DEL FORMATO
105 SP$=CHR$(29)
110 FOR I=1 TO 8
120 READ M$
140 NEXT I
150 PRINT#2,"AAAAAAAAAAAAAA"
160 FOR I=1 TO 5
170 PRINT#1,M$(I)
180 NEXT I
190 PRINT#2,"AAAAAAAAAAA   AAAAAAA   AAA"
200 PRINT#1,M$(6)SP$M$(7)SP$M$(8)
210 CLOSE 1
220 CLOSE 2
230 STOP
```

```
1  MARY PERKINS
2  35 WEST ST.
3  BERKELEY
4  CALIFORNIA
5  94705
6  345-67-8910   SPONSOR   AXC
```

Questo programma è una variante di STR.FORM.PRINT1. Alle due PRINT delle linee 150 e 190 è stato aggiunto un campo numerico seguito da tre spazi vuoti. In questo campo sarà inserito un valore intero progressivo prelevato dall'indice del ciclo FOR NEXT di stampa.

PRINTDATE è un altro programma molto interessante. Esso riceve dalla tastiera in forma numerica il giorno, il mese e l'anno di una data e poi la stampa con un

trattino di separazione tra il giorno e il mese e tra il mese e l'anno. Ecco il suo listato ed un esempio di esecuzione:

```
10 REM PROGRAMMA "PRINTDATE"
20 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
30 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
35 PRINT#2,"AAAAA   99A99A99"
40 PRINT"□□□"
50 INPUT "BATTERE IL GIORNO:";D
60 INPUT "BATTERE IL MESE  :";M
70 INPUT "BATTERE L'ANNO   :";Y
90 SP$=CHR$(29)
100 PRINT#1,"DATA:"SP$,D,"-"SP$,M,"-"SP$,Y
110 PRINT"UN'ALTRA DATA? BATTERE Y PER SI O N PER NO ";
120 GET YN$: IF YN$="" THEN 120
130 IF YN$="N" THEN PRINT YN$: STOP
140 IF YN$<>"Y" THEN 120
150 GOTO 40

DATA:   21-12-74
DATA:   20- 3-76
DATA:   15- 9-82
```

Attenzione però che PRINTDATE non fa un controllo della validità dei valori che gli date in ingresso; esso cura unicamente l'aspetto della tabulazione sulla stampante.

Inserimento di lettere nella stampa con formato

La stringa che rappresenta un formato può contenere dei caratteri letterali. Tali lettere vengono poi stampate così come sono poste nel formato e non hanno alcun valore per il comando PRINT # successivo. Queste lettere devono essere precedute dal comando REVERSE ON (RVS) e sono poi stampate normalmente. Di conseguenza non potete mai stampare caratteri con il campo invertito.

Il programma PRINTDATEL1 fa un uso molto semplice di caratteri letterali. Esso fa apparire i trattini posti tra il giorno, il mese e l'anno con la tecnica che abbiamo appena descritto. Per ottenere il programma PRINTDATEL1 modificate le linee 80 e 100 del programma PRINTDATE. Ambedue questi programmi danno lo stesso tabulato in uscita.

```
 10 REM PROGRAMMA "PRINTDATEL1"
 20 OPEN 1,4,1: REM USCITA DEI DATI TRAMITE FILE LOGICO 1
 30 OPEN 2,4,2: REM USCITA DEL FORMATO TRAMITE FILE LOGICO 2
 35 PRINT#2,"AAAAA   99□-99□-99"
 40 PRINT"□□□"
 50 INPUT "BATTERE IL GIORNO:";D
 60 INPUT "BATTERE IL MESE  :";M
 70 INPUT "BATTERE L'ANNO   :";Y
 90 SP$=CHR$(29)
 100 PRINT#1,"DATA:"SP$,D,M,Y
 110 PRINT"UN'ALTRA DATA? BATTERE Y PER SI O N PER NO ";
 120 GET YN$: IF YN$="" THEN 120
 130 IF YN$="N" THEN PRINT YN$: STOP
 140 IF YN$<>"Y" THEN 120
 150 GOTO 40
READY.

DATA:   10- 1-82
DATA:    8-12-81
DATA:    2- 6-74
```

Mediante un uso appropriato dei caratteri letterali potete creare dei moduli, ma sempre facendo riferimento allo stesso insieme di caratteri del testo. Ricordate inoltre che le stampanti riconoscono lo stesso insieme di caratteri dei PET.

CARATTERI SPECIALI DI CONTROLLO DELLA STAMPANTE

Nella Tabella 6-4 sono riportati i caratteri speciali di controllo delle stampanti CBM.

Tali caratteri speciali di controllo sono trasmessi alla stampante ponendo l'indirizzo secondario 0 o 1. Essi non fanno parte del formato che è invece trasmesso con l'indirizzo secondario 2.

I caratteri speciali possono essere usati sia quando si fanno le stampe con formato sia quando non si usa il formato.

I primi due caratteri speciali della Tabella 6-4 CHR$(29) e CHR$(160), che abbiamo già incontrato, devono essere usati solo in stampe con formato. Se usati diversamente sono ignorati.

I codici della Tabella 6-4, indicati come opzionali, possono essere usati sia con stampe con formato che senza formato e danno gli stessi risultati.

Stampa con caratteri allargati

Le stampanti CBM generano i caratteri mediante una matrice a punti le cui dimensioni sono sette punti in altezza e sei punti in larghezza. **È possibile però stampare i caratteri con larghezza doppia e cioè 12 punti in larghezza e sette in altezza anteponendo alle variabili della lista della PRINT # il comando CHR$(1).** Il comando CHR$(1) raddoppia la larghezza di tutti i caratteri che lo seguono. È possibile quindi raddoppiare più volte la larghezza dei caratteri inserendo ripetutamente CHR$(1). Si possono ottenere così caratteri con larghezza di 12, 24, 48, ecc. punti.

Per illustrarvi come si possono generare caratteri allargati, aggiungete questa linea al programma STR.FORM.PRINT1:

```
125 M$(I)=CHR$(1)+M$(I)
```

Nel nuovo tabulato vedrete che la prima colonna comprendente il nome, l'indirizzo e il numero di matricola sarà stampata con caratteri di larghezza doppia. La seconda colonna avrà invece caratteri con larghezza quadrupla e l'ultima con larghezza otto volte superiore:

```
MARY PERKINS
35 WEST ST.
BERKELEY
CALIFORNIA
94705
345-67-8910     SPONSOR        FINO
```

Infatti alla linea 125 si è aggiunto il comando CHR$(1) ad ogni stringa M$(I) per

cui con la PRINT # della linea 170 si ha un raddoppio della larghezza dei caratteri e con la PRINT # della linea 200 si hanno effetti multipli.

Non è necessario concatenare sempre CHR$(1) alle stringhe ma **potete anche inserirlo direttamente nella PRINT #** senza però l'uso di virgole. Per esempio la linea 200 del programma STR.FORM.PRINT1 può essere sostituita con queste due linee:

```
195 E$=CHR$(1)
200 PRINT#1,E$M$(6)SP$E$M$(7)SP$E$M$(8)
```

In questo caso il nome e l'indirizzo sono tabulati con caratteri standard mentre invece l'ultima riga è stampata con i caratteri allargati.

```
MARY PERKINS
35 WEST ST.
BERKELEY
CALIFORNIA
94705
345-67-8910     SPONSOR     AKC
```

Anche le variabili numeriche possono essere stampate con caratteri allargati, ma in questo caso esse devono essere separate da virgole nella lista della PRINT #. Per darvi una dimostrazione pratica modificate le linee 190 e 200 del programma STR.FORM.PRINT1 nel modo seguente:

```
190 PRINT#2,"AAAAAAAAAAA   99999  AAAAAAA"
195 E$=CHR$(1)
196 N=12345
200 PRINT#1,E$M$(6)SP$,N,E$M$(7)
```

Il formato della linea 190 prevede ora due campi di stringa e uno numerico. Nella lista della PRINT # della linea 200 comapiono le due stringhe M$(6) e M$(7) e la nuova variabile numerica N il cui valore è stato definito come 12345 alla linea 196. **Notate che la variabile numerica è separata da virgole. Le variabili di stringa non richiedono invece di essere separate da virgole.** Questa differenza, tra variabili di stringa e numeriche, è molto importante e va tenuta presente quando si mescolano i due tipi di variabili in una unica PRINT # con caratteri allargati. Qui di seguito vi diamo un esempio di tabulato del programma STR.FORM.PRINT1 con le modifiche appena suggerite:

```
MARY PERKINS
35 WEST ST.
BERKELEY
CALIFORNIA
94705
345-67-8910     12345   SPONSOR
```

Per annullare gli effetti del comando CHR$(1) dovete usare il comando CHR$(129). Questo comando ripristina la stampa dei caratteri standard sino a che non sia dato un altro comando CHR$(1).

Stampa di caratteri con campo invertito

Caratteri con campo invertito possono essere inclusi in una istruzione PRINT # mediante il tasto RVS ON. Vi consigliamo però di non stampare più di quattro o cinque linee consecutive con i caratteri invertiti in quanto ciò porta ad un rapido deterioramento della testina della stampante.

Stampa dei caratteri di controllo

Per stampare le virgolette dovete usare la funzione CHR$(34). Se poi inserite in una istruzione di stampa un numero dispari di CHR$(34), allora la stampante farà apparire tutti i caratteri di controllo con la loro rappresentazione grafica.

Ovviamente l'unica applicazione di questo comando la si ha quando dovete listare un programma che contenga caratteri di controllo che altrimenti non sarebbero stampati.

IMPAGINAZIONE CON FORMATO

Numero di righe per pagina

Se non viene dato alcun comando speciale le stampanti CBM stampano pagine di lunghezza indefinita.

Per avere invece una lunghezza delle pagine definita dovete dare il comando CHR$(147). In questo caso la stampante assume che ogni pagina sia lunga 66 righe per cui ne stampa 60 e ne lascia libere 6 per passare alla nuova pagina. Qui di seguito vi diamo un breve esempio di programma che tiene conto della successione delle pagine:

```
10 REM PROGRAMMA "PAGING"
15 REM PROVA DI IMPAGINAZIONE
20 OPEN 1,4: REM STAMPA SENZA FORMATO
30 REM ABILITAZIONE IMPAGINAZIONE
40 PRINT#1,CHR$(147)
50 FOR I=1 TO 100
60 PRINT#1,I,"ABCDEFG"
70 NEXT I
80 CLOSE 1
90 STOP
```

Il numero di righe per pagina può essere cambiato purchè abbiate già abilitato il controllo di impaginazione. Per fare questo dovete aprire un nuovo file logico con numero di unità fisica 4 e indirizzo secondario 3. Poi dovete dare un comando PRINT # con il numero di righe voluto. Attenzione però che la stampante aggiunge automaticamente le sei righe per la spaziatura tra una pagina e l'altra. Cioè il numero da voi indicato rappresenta il numero effettivo di righe per pagina stampa-

bili. Ecco un esempio di programma, denominato PAGINGL25, che stampa 25 righe per pagina:

```
10 REM PROGRAMMA "PAGINGL25"
15 REM PROVA DI IMPAGINAZIONE
20 OPEN 1,4: REM STAMPA SENZA FORMATO
25 OPEN 3,4,3: REM APRE FILE PER DARE NUMERO LINEE/PAGINA
30 REM ABILITAZIONE IMPAGINAZIONE
40 PRINT#1,CHR$(147)
45 PRINT#3,25: REM IMPONE 25 LINEE/PAGINA
50 FOR I=1 TO 100
60 PRINT#1,I,"ABCDEFG"
70 NEXT I
80 CLOSE 1
85 CLOSE 3
90 STOP
```

L'istruzione PRINT # della linea 45 specifica 25 righe per pagina. Il file logico 3 è aperto alla linea 25.

Se volete potete cambiare, in più punti del programma, la lunghezza delle pagine. Infatti basta porre altre PRINT # 3 con il nuovo numero di righe per pagina. Attenzione però che questo numero diventa effettivo solo all'inizio di una nuova pagina.

Provate per esercizio ad inserire questa nuova linea nel programma PAGINGL25:

```
55 IF I=23 THEN PRINT#3,10
```

ed eseguite il programma due volte. Durante la prima esecuzione viene stampata una pagina di 25 righe e poi un certo numero di pagine di 10 righe. Quando poi eseguirete per la seconda volta il programma non otterrete gli stessi risultati! La prima pagina sarà di 10 righe, la seconda di 25 righe e tutte le altre di 10 righe ancora. Infatti all'inizio della seconda esecuzione la stampante "ricorderà" ancora il valore di righe per pagina dell'esecuzione precedente.

Inizio pagina

Mentre il controllo di impaginazione è abilitato potete dare il comando CHR$(19) che fa saltare la stampa all'inizio della pagina successiva. Ricordatevi che è una buona norma terminare la scrittura di una pagina sempre con questo comando CHR$(19) per essere sicuri che la stampa della nuova pagina inizi effettivamente sul foglio successivo.

Spaziatura tra le righe (Modello 2022)

La stampante modello 2022 permette di cambiare la spaziatura tra le righe.

Per capire come questo possa avvenire pensate ad un asse verticale del foglio. Ogni pollice di questo asse è ripartito in 144 gradini e ogni riga ne occupa 24. Così

per ogni pollice verticale vengono stampate 6 righe. **Nel modello 2022 è possibile cambiare il numero di righe stampate per pollice.** Per fare questo dovete aprire un file logico con numero di unità fisica 4 e indirizzo secondario 6. Mediante una PRINT# trasferite la funzione CHR$ con il nuovo numero di gradini per riga posto come suo argomento.

Supponiamo per esempio che vogliate 8 righe per pollice; il numero di gradini per riga diviene 144/8 = 18. Le istruzioni necessarie sono allora:

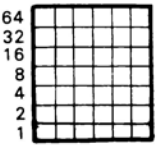
```
10 OPEN 6,4,6
20 PRINT#6,CHR$(18)
```

Se ora ponete queste istruzioni nel programma PAGINGL25 vedrete che le righe sono stampate senza spaziatura. Questo perchè l'altezza dei caratteri rimane sempre la stessa e se aumenta il numero di righe per pollice, ciò va a discapito della spaziatura tra le righe. Provate a cambiare il valore posto come argomento di CHR$ e vedere che cosa succede. In alcuni casi le righe saranno ben spaziate e in altri saranno addirittura sovrapposte.

DEFINIZIONE DI CARATTERI PERSONALIZZATI

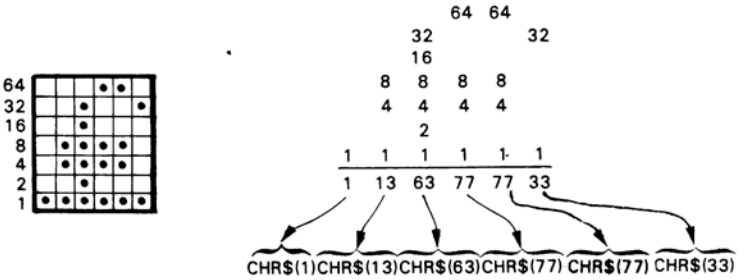
Le stampanti CBM vi permettono di definire o disegnare nuovi caratteri a seconda delle vostre necessità.

Tutti i caratteri stampabili sono generati da una matrice a 7x6 punti come la seguente:



Ogni riga di questa matrice è rappresentata da un numero compreso tra 1 e 64. La riga più in alto ha il numero 64 quella più in basso 1.

Costruite allora il vostro nuovo carattere come un insieme di punti della matrice 7x6. Ecco l'esempio per generare il simbolo della sterlina:



A questo punto bisogna convertire il nostro disegno in 6 numeri che rappresentino i punti su ciascuna colonna della matrice nell'ordine da sinistra verso destra.

Per ottenere questi numeri dovete fare la somma verticale dei valori di riga di ciascun punto, come facilmente potete capire dal disegno di sopra. Questi sei valori diventano poi i sei argomenti di sei funzioni CHR$. **In definitiva il nuovo carattere diventa una stringa di sei caratteri ognuno definito dalla funzione CHR$.** Nel caso della sterlina i sei caratteri sono: CHR$(1), CHR$(13), CHR$(63), CHR$(77), CHR$(77) e CHR$(33).

A questo punto la stringa del nuovo carattere deve essere trasmessa alla stampante mediante una istruzione PRINT # che faccia riferimento ad un file logico aperto con numero di unità fisica 4 e indirizzo secondario 5. La stampante però non stamperà il nuovo carattere, ma lo memorizzerà. Per ottenere la stampa dovrete dare il comando CHR$(254) posto in una PRINT #. **Da questo si deduce che si può utilizzare solo un carattere speciale alla volta.** Per usare altri caratteri voi li potete definire tutti inizialmente, ma usare solo uno alla volta trasmettendo prima la sua stringa alla stampante con l'indirizzo secondario 5.

Per fare un esempio concreto vi diamo qui di seguito il programma STERLINA che genera un tabulato con 10 simboli £ posti in colonna:

```
10 REM  PROGRAMMA "STERLINA"
15 REM
20 REM ESEMPIO DI GENERAZIONE DI
25 REM  UN CARATTERE SPECIALE
26 REM
30 DATA 1,13,63,77,77,33
35 EP$=""
40 OPEN 1,4: REM APERTURA STAMPANTE
50 OPEN 5,4,5:REM APERTURA FILE GENERAZ. CARAT.
60 FOR I=1 TO 6
70 READ EP
80 EP$=EP$+CHR$(EP)
90 NEXT I
95 PRINT#5,EP$
100 FOR I=1 TO 10
110 PRINT#1,CHR$(254)
120 NEXT I
130 CLOSE 1
140 CLOSE 5
150 STOP
```

Esaminiamo ora il funzionamento di questo programma. L'istruzione DATA alla linea 30 fornisce la codifica dei punti del nuovo simbolo come abbiamo descritto più sopra. Mediante il ciclo FOR NEXT dalla linea 60 alla 90 viene costruita la stringa EP$ che contiene il nuovo carattere. Alla linea 95 la stringa EP$ è trasferita alla stampante. Il simbolo di sterlina viene poi stampato alla linea 110 mediante il comando CHR$(254).

Uso di caratteri speciali per stampare simboli monetari diversi dal dollaro

È molto probabile che dobbiate usare simboli monetari diversi da quello del dollaro. Ciò si può facilmente ottenere facendo uso di programmi analoghi a

STERLINA inseriti in programmi di stampa con formato.

Ecco per esempio il programma STERLINAVAL che stampa importi in sterline:

```
10 REM  PROGRAMMA "STERLINAVAL"
15 REM
20 REM   STAMPA DI VALORI IN
25 REM    STERLINE INGLESI
26 REM (CREAZIONE CAR. STERLINA)
27 REM
40 DATA 1,13,63,77,77,33
50 OPEN 5,4,5
60 EP$=""
70 FOR I=1 TO 6
80 READ EP
90 EP$=EP$+CHR$(EP)
100 NEXT I
110 PRINT#5,EP$
120 OPEN 1,4,1: REM STAMPA CON FORMATO
130 OPEN 2,4,2
140 REM USCITA DATI CON CAR. STERLINA
150 PRINT#2,"AAAAAAA   A999999.99-"
160 INPUT "DARE L'IMPORTO:";N
170 PRINT#1,"VALORE="CHR$(29)CHR$(254)CHR$(29),N
180 CLOSE 1
190 CLOSE 2
200 CLOSE 5
210 STOP

VALORE=   £  1234.56

VALORE=   £   400.00-
```

Il simbolo di sterlina è creato alle linee da 40 a 110 che sono analoghe a quelle del programma STERLINA.

Le istruzioni OPEN delle linee 120 e 130 aprono i file logici 1 e 2 per la stampa con formato. Il formato è definito alla linea 150 e prevede un campo di stringa con 6 caratteri, tre spazi bianchi, un campo di stringa, con un solo carattere, e un campo numerico con il segno in coda.

Alla linea 160 viene richiesto un valore numerico N in ingresso che viene poi stampato alla linea 170.

Nell'istruzione PRINT # della linea 170 troviamo un testo "VALORE =" che viene stampato con il primo campo del formato seguito dal separatore obbligatorio CHR$(29). Poi dopo i tre spazi bianchi troviamo CHR$(254), che fa stampare il segno di sterlina, seguito anche lui da CHR$(29) e infine la variabile numerica N preceduta dalla virgola.

MESSAGGI DIAGNOSTICI DELLA STAMPANTE

Se vi accorgete di ottenere tabulati non corretti, potete abilitare la stampa di messaggi diagnostici. Per fare questo aprite un file logico con numero di unità fisica 4 e indirizzo secondario 4. Ciò sarà sufficiente per darvi gli avvisi di errore senza che dobbiate dare alcuna altra istruzione di uscita.

Ovviamente i programmi definitivi non devono contenere i comandi per la stampa dei messaggi diagnostici.

Per fare un esempio caricate il programma STR.FORM.PRINT1 e cambiate una delle A del formato alla linea 190 con una Q per esempio. Aggiungete quindi l'istruzione di abilitazione della diagnostica:

```
85 OPEN 4,4,4
```

Quando porrete in esecuzione il programma otterrete un messaggio di errore simile a questo:

```
MARY PERKINS
35 WEST ST.
BERKELEY
CALIFORNIA
94705
345-6
AAAAAQAAAAA   AAAAAAA   AAA
     ↑
*****BAD FORMAT*****
PONSOR AXC
```

CAPITOLO 7

INFORMAZIONI SUL SISTEMA CBM

ORGANIZZAZIONE DEL SISTEMA CBM

I calcolatori CBM impiegano il microprocessore 6502. Nel capitolo 2 abbiamo descritto le unità fisiche esterne: display video, unità a cassette, unità a dischetti e stampanti. Le tre porte di ingresso/uscita sono interfacciate tramite un blocco di memoria a mappa di 2K. L'organizzazione del sistema CBM è illustrata nella Figura 7-1. Nel caso dei calcolatori PET 8K la prima unità a cassette è collegata direttamente al blocco I/O mentre la seconda cassetta viene collegata all'apposita interfaccia per cassette. Nel caso invece dei calcolatori PET 16K/32K la prima unità è collegata tramite quest'ultima interfaccia e la seconda tramite l'interfaccia IEEE 488. I sei blocchi ROM, RAM e I/O sono ottenuti dai 65 Kbyte di memoria totale disponibile (ricordiamo che 1K = 1024).

In Tabella 7-1 è illustrata la ripartizione della memoria in blocchi di 4K. Ogni porzione di memoria verrà descritta nei dettagli in seguito.

Tabella 7-1: Impiego della memoria in blocchi di 4K

Blocchi	Tipo di memoria	Indirizzo iniziale		Descrizione
		Decimale	Esadecimale	
0	RAM	0	0000	Memoria di lavoro, inizio del testo
1	RAM	4096	1000	Memoria del testo e delle variabili (solo per 8 K)
2	—	8192	2000	Espansione della RAM
3	—	12288	3000	
4	—	16384	4000	
5	—	20480	5000	
6	—	24576	6000	
7	—	28672	7000	
8	RAM	32768	8000	Memoria dello schermo (e dell'I/O solo in BASIC 4.0)
9	ROM	36864	9000	Espansione della ROM
10	ROM	40960	A000	
11	ROM	45056	B000	Inizio del BASIC 4.0
12	ROM	49152	C000	BASIC (Interprete istruzioni principali)
13	ROM	53248	D000	BASIC (Software matematico)
14	ROM	57344	E000	Editore dello schermo (2 K)
	I/O	59392	E800	Memoria di I/O (2 K)
15	ROM	61440	F000	Sistema Operativo (OS)

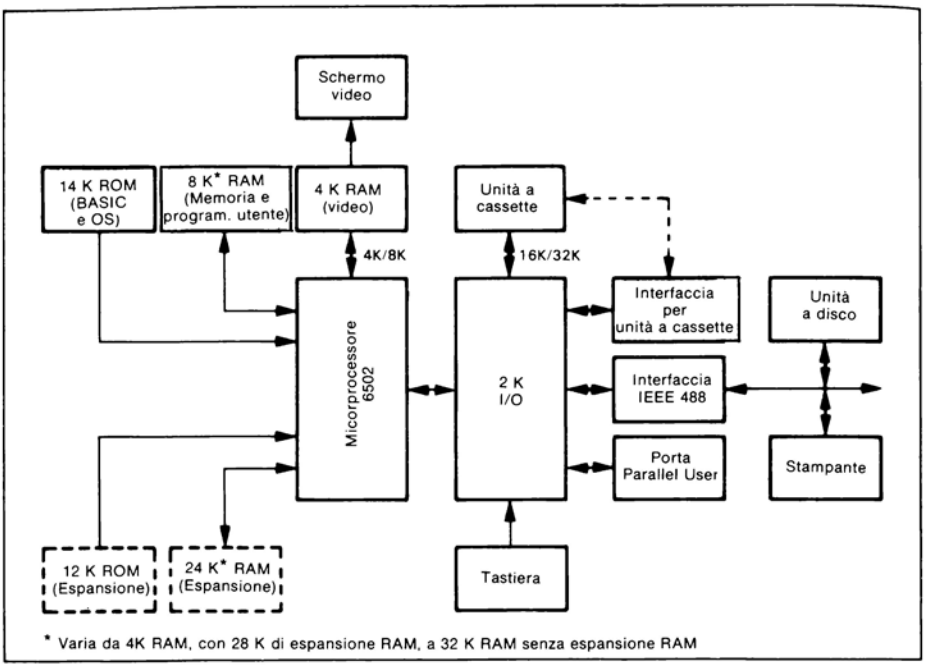


Figure 7-1. Schema a blocchi del PET.

Indirizzi 0 — 8191: 8K RAM (Memoria e Programmi Utente)

Il primo blocco di memoria RAM è ripartito tra: memoria di lavoro, "stack", memorie buffer per i nastri e memoria per i programmi dell'utente.

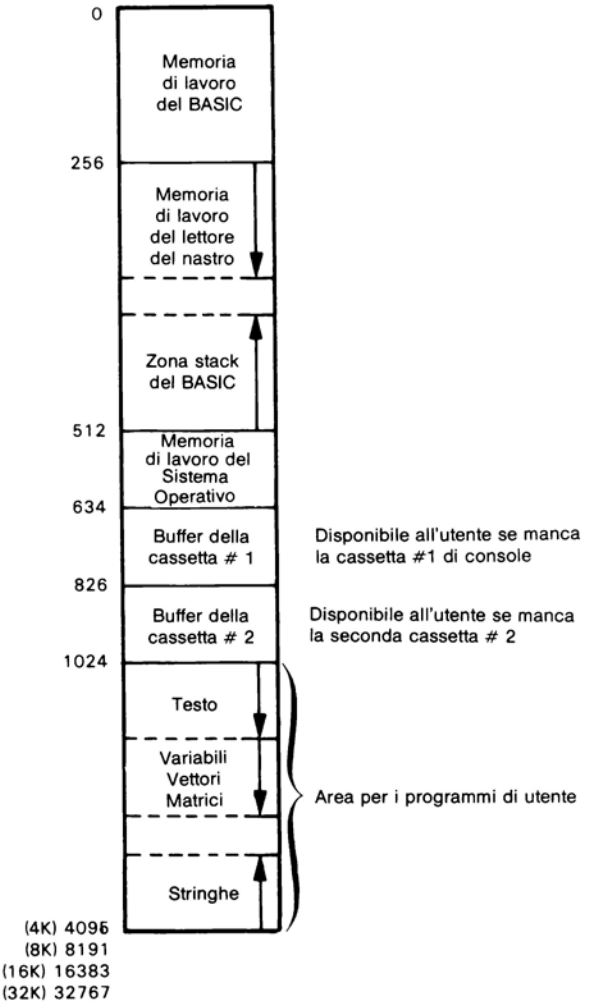
La memoria RAM può avere una dimensione di 8K (indirizzi da 0 a 8191), di 16K (indirizzi da 0 a 16383) oppre di 32K (indirizzi da 0 a 32767). L'impiego del primo K di memoria è sempre fisso (da 0 a 1024); i rimanenti sono a disposizione dell'utente per cui più grande è la memoria, maggior spazio è riservato ai programmi.

Gli indirizzi da 0 a 255 sono usati dall'interprete BASIC come memoria di lavoro. La descrizione dettagliata di questa zona è riportata nella Appendice F.

La porzione di memoria dall'indirizzo 256 all'indirizzo 511 è usata principalmente dallo Stack BASIC ed è ripartita dinamicamente in due zone. La prima inizia dal basso (256) e procede verso l'alto con la routine del lettore di nastro per la correzione degli errori e con un eventuale buffer di espansione del BASIC. La seconda zona inizia dall'alto (511), prosegue verso il basso ed è occupata dallo stack. Se le due zone collidono viene dato l'avviso: OUT OF MEMORY.

Gli indirizzi da 512 a 633 sono usati dal "Sistema Operativo" (OS) come memoria di lavoro. Questa zona è descritta nell'Appendice F.

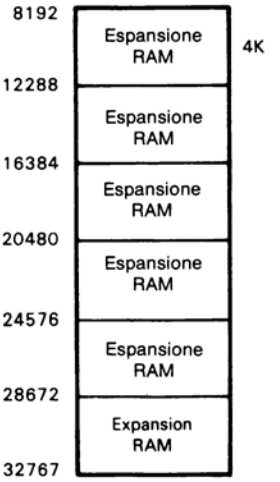
La zona da 634 a 825 costituisce il buffer di 192 byte per la prima unità a cassette. Mentre la zona da 826 a 1023 costituisce l'eventuale secondo buffer di 192 byte per un'altra unità a cassette. Questi buffer possono essere usati per contenere programmi scritti dall'utente in Assembler nel caso che una o ambedue le unità a nastro non siano usate.



La memoria RAM dall'indirizzo 1024 sino alla fine è usata per contenere i programmi e le variabili dell'utente. Un programma inizia all'indirizzo 1024 e prosegue verso l'alto. Le variabili, con e senza indici, iniziano alla fine del programma. Le stringhe invece sono memorizzate a partire dalla fine della memoria e proseguono verso il basso. Ovviamente può avvenire un errore di superamento della memoria OUT OF MEMORY se il puntatore delle stringhe, mentre scende verso il basso, incontra l'altro puntatore che sta salendo.

Indirizzi 8192 — 32767: Espansione di 24K della RAM

Gli indirizzi di memoria da 8192 a 32767 sono usati nel caso di espansione della RAM a 32K.



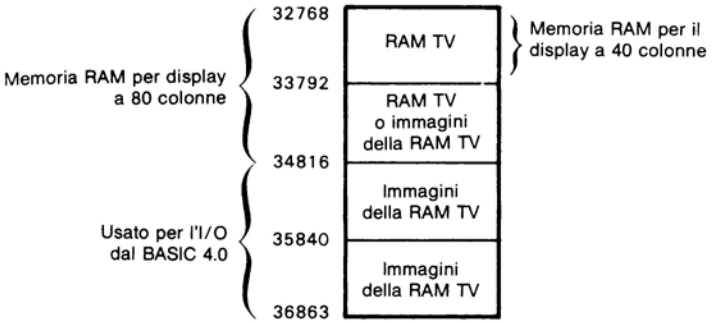
Il totale di 32K di memoria può essere costituito in vario modo tra la RAM attiva e quella di espansione:

RAM attiva	RAM di espansione
4K (0—4095)	28K (4096—32767)
8K (0—8191)	24K (8192—32767)
16K (0—16383)	

Indirizzi 32768 — 36863: Memoria RAM di 4K per l'unità video

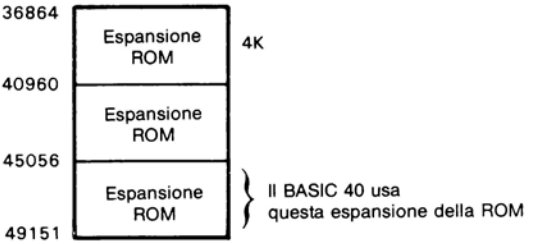
I primi mille indirizzi di questo blocco, da 32768 a 33767 sono riservati alla memoria dello schermo. Un comando POKE riferito a questi indirizzi fa visualizza-

re un carattere nella corrispondente posizione sullo schermo.



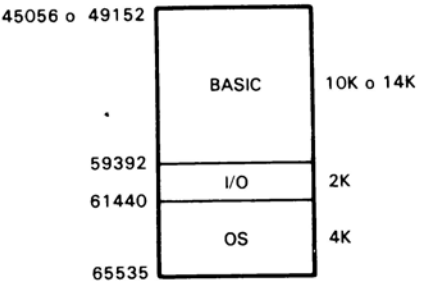
Indirizzi 36864 — 49151: Espansione di 12K della ROM

Gli indirizzi da 36864 a 49151 sono riservati per la espansione opzionale della ROM a 26K.



Indirizzi 49152 — 65535: 14K di ROM e 2K per I/O

Gli indirizzi da 49152 (45056 nel caso invece del BASIC 4.0) sino a 59391 e da 61440 a 65535 sono riservati all'interprete BASIC e alla diagnostica del Sistema Operativo (OS). Da 59392 a 61439 è presente la memoria a mappa per l'I/O.



L'indirizzo 65535 è l'ultimo della memoria dei calcolatori CBM.

MAPPA DELLA MEMORIA

Nell'Appendice D potete trovare le mappe dettagliate della memoria corrispondenti alle diverse versioni del BASIC CBM. Nella Tabella D-1 è descritta la revisione livello 2 della ROM usata nei primi calcolatori PET. Nella Tabella D-2 è descritta la revisione livello 3 della ROM usata nel BASIC < 3.0. E infine nella Tabella D-3 è descritta la più recente mappa di memoria dl BASIC 4.0.

Nelle Tabella D-1 e D-2 gli indirizzi di memoria sono dati sia in forma decimale che esadecimale. **Nelle istruzioni PEEK e POOK dovete però usare il valore decimale.** Nelle stesse tabelle, trovate anche il corrispondente contenuto di memoria sia in forma decimale che esadecimale.

Il contenuto rappresenta quello che potreste trovare in quell'indirizzo con l'istruzione PEEK salvo che non si tratti di un puntatore. I contenuti sono sempre dei byte con valore da 0 a 255 (O-FF esadecimale). **Un puntatore è un indirizzo a due byte, con valore tra 0 e 65535 (FFFF esadecimale), che è memorizzato nei calcolatori CBM nell'ordine byte-basso, byte-alto.** Tutti gli indirizzi a due byte nelle tabelle hanno un contenuto espresso nell'ordine basso-alto. Consideriamo per esempio il primo indirizzo nella tabella:

Indirizzo di memoria		Contenuto della posizione		Descrizione
Decimale	Esadecimale	Decimale	Esadecimale	
1-2	0001-0002	826	033A	Vettore di salto all'indirizzo di utente

Se date il comando PEEK l'indirizzo a 16 bit sarà presente in due parti; prima il byte di ordine basso:

```
?PEEK(1)
58
```

e poi il byte di ordine alto:

```
?PEEK(2)
3
```

Per convertire i due valori ottenuti in un unico indirizzo, potete convertirli separatamente in esadecimale e quindi da esadecimale in decimale:

Basso	Alto		Indirizzo
$58_{10}=3A_{16}$	$3_{10}=03_{16}$	$\longrightarrow$	$033A_{16}=826_{10}$

Fate molta attenzione che il valore 033A è ottenuto dal byte basso 3A e dal byte alto 03.

Potete anche moltiplicare il byte-alto per 256 e sommarlo al byte-basso. Per esempio così:

```
?PEEK(1)*256+PEEK(2)
826
```

Viceversa per convertire un indirizzo di 16-bit in due byte separati, per inserirli nell'istruzione POKE, convertite il valore decimale in esadecimale. Poi separate i due byte e convertiteli in decimale. Per esempio convertiamo l'indirizzo 59409:

		Alto	Basso
$59409_{10} = E811_{16}$	→	$E8_{16} = 232_{10}$	$11_{16} = 17_{10}$

È possibile anche convertire questo numero, con la normale aritmetica decimale, dapprima dividendolo per 256 e trascurando la parte frazionaria:

59409/256 = 232.06641 232 **Alto**

e poi sottraendo al numero originario il valore alto, appena trovato, moltiplicato per 256:

232 ★ 256 = 59392
59409 — 59392 = 17 17 **Basso**

Nella colonna DESCRIZIONE della Tabella D-1 è riportata una breve descrizione dell'uso delle posizioni. Nel caso che una posizione abbia più impieghi, nella descrizione è riportato solo il principale. Pur non essendo completa la tabella illustra la struttura fondamentale della memoria CBM.

Nella Tabella D-3 si confronta la mappa di memoria del BASIC 4.0 con quella del BASIC 3.0 presentata nella Tabella D-2. La DESCRIZIONE delle posizioni è quella normalmente usata dalla Commodore; l'ETICHETTA indica l'etichetta del linguaggio Assembler normalmente assegnata per le posizioni dalla Commodore. La colonna BASIC 4.0 riporta gli indirizzi in forma esadecimale equivalenti a quelli della colonna BASIC 3.0. Per trovare una posizione in BASIC 4.0 cercate prima nella colonna della Tabella D-2; ritrovate poi il corrispondente indirizzo nella colonna BASIC 3.0 della Tabella D-3 e comparatelo con quello adiacente della colonna BASIC 4.0.

A differenza dei primi due indirizzi iniziali della Tabella D-3 che rappresentano in effetti l'indirizzo di memoria 0000, tutti gli altri indirizzi 0000 identificano valori che non esistono in una delle due versioni in BASIC. Per esempio se vedete un indirizzo nella prima colonna a cui corrisponde un valore 0000 nella seconda colonna, questo significa che non esiste una posizione equivalente in BASIC 4.0. Viceversa se il valore 0000 è presente nella prima colonna e non nella seconda, significa che non esiste questa posizione in BASIC 3.0.

L'INTERPRETE BASIC CBM

L'interprete BASIC CBM esegue un programma di utente decodificando ogni sua linea. Le linee del programma sorgente sono memorizzate in forma compatta. Quando battete una linea sulla tastiera siete sotto il controllo dell'editor sino a che non premete il tasto RETURN. Le linee di programma sono memorizzate con il loro numero in

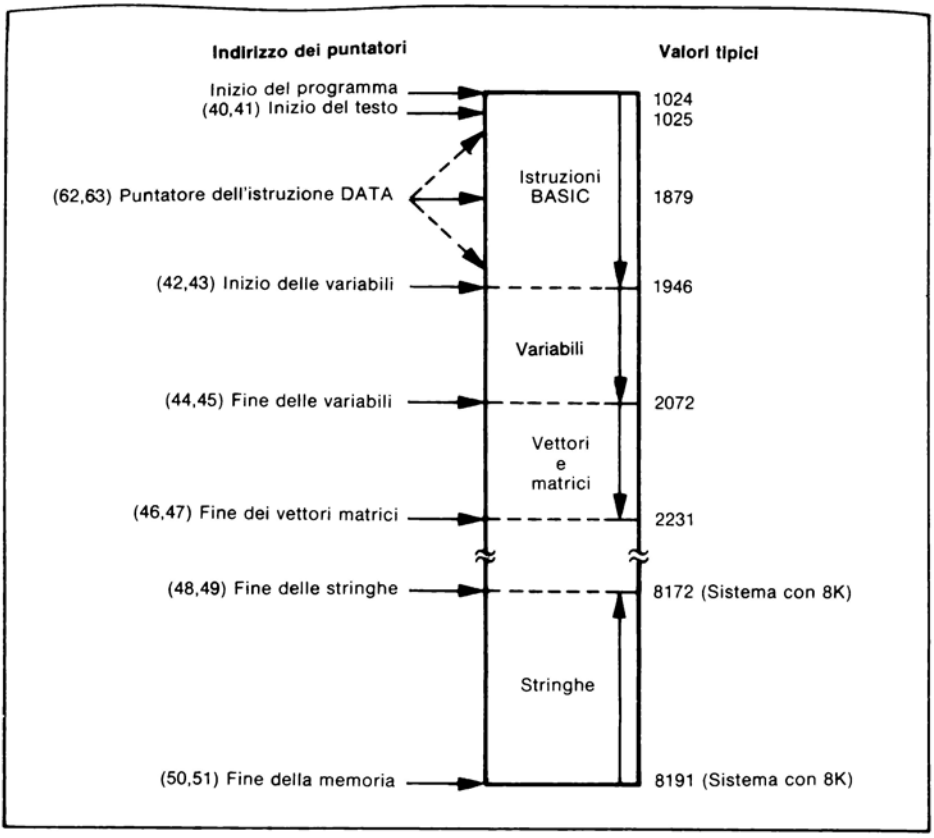


Figura 7-2: Puntatori principali dell'area programmi di utente

ordine crescente. Quando confermate una nuova linea l'interprete ricerca in memoria lo stesso numero di linea; se lo trova sostituisce la linea, diversamente inserisce la nuova linea.

Le linee di programma sono memorizzate a partire dall'indirizzo 1024 nella zona riservata ai programmi di utente. Le variabili sono memorizzate dopo le linee di programma. Tutte e due le aree iniziano agli indirizzi più bassi e proseguono verso l'alto. Le stringhe invece iniziano all'indirizzo più alto e proseguono verso il basso. L'interprete BASIC sovrintende alle aree, muovendole se necessario e aggiustando i puntatori per le inserzioni e le cancellazioni. Otto copie di posizioni di memoria contengono i puntatori per individuare i punti di divisione dell'area di utente. Essi sono indicati nella Figura 7-2 e nelle tavole dell'Appendice D.

I formati con cui le istruzioni BASIC, le variabili, i vettori, le matrici e le stringhe sono memorizzati nelle rispettive aree saranno discussi in seguito.

MEMORIZZAZIONE DELLE ISTRUZIONI BASIC

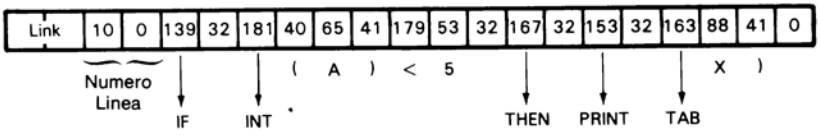
Le istruzioni BASIC sono memorizzate con il formato presentato in Figura 7-3.

La posizione di memoria 1024 contiene sempre un byte zero. I successivi due byte contengono un puntatore puntato all'inizio della prima istruzione BASIC. Il puntatore, come ogni altro indirizzo, è memorizzato in due byte con ordine basso e alto. Il puntatore ha una funzione di collegamento ("link") all'istruzione successiva tramite il suo corrispondente link. **Un indirizzo zero di link indica la fine del testo**; cioè che non ci sono altre istruzioni successive. Le istruzioni BASIC sono sempre memorizzate in ordine crescente di numero di linea, anche se non sarebbe necessario per la presenza dei link. Il compito principale dei puntatori link è quello di selezionare rapidamente i numeri di linea.

Subito dopo l'indirizzo di link viene il numero di linea dell'istruzione memorizzato in due byte con ordine basso e alto. I numeri di linea possono andare da 1 (memorizzato come 1 o 0) sino a 63999 (memorizzato come 255 e 249).

Dopo il numero di linea inizia il testo vero e proprio dell'istruzione. Le parole chiave sono le parole riservate (vedi Tabella 4-4) e gli operatori (vedi Tabella 4-2). Le parole riservate e gli operatori logici sono memorizzati in forma compatta. Ogni parola chiave è contrassegnata da un byte. **Tutte le parole chiave sono codificate con il bit di ordine superiore pari a 1.** Gli altri elementi del testo BASIC sono codificati con il codice ASCII e comprendono le costanti, le variabili, le variabili con indici, le stringhe e i simboli speciali diversi dagli operatori. Essi vengono codificati così come appaiono nel testo originale. La Tabella A-1 riporta i byte in codice di tutti i valori tra 0 e 255 che possono apparire in un testo BASIC compatto. I codici sono sempre interpretati secondo questa tabella eccetto quando si tratta di una stringa e cioè dopo un numero dispari di virgolette. **All'interno di una stringa si applica unicamente il codice ASCII riportato in Tabella A-4.**

Notate che la parentesi di sinistra è già contenuta e memorizzata nel codice delle funzioni TAB e SPC, a differenza delle altre funzioni che usano un byte separato per codificarla. Per esempio qui di seguito vi diamo la codifica in byte (decimali) di una istruzione BASIC:



Gli operatori (+, −, ★, /, <, =, >, AND, OR e NOT) vengono codificati come le parole chiavi (cioè il bit di ordine superiore è uguale a 1) giacchè essi "guidano" l'interprete BASIC come se fossero parole riservate. Per esempio 179 viene attribuito a < mentre il suo codice ASCII è 60. Il valore 60 per < appare solo all'interno di una stringa.

Tutti gli spazi della linea sorgente sono memorizzati ad eccezione di quelli tra il numero di linea e la prima parola. Questi ultimi verranno poi ricostruiti se si dovesse richiedere la ristampa del testo originale. Voi potete risparmiare la memoria eliminando gli spazi del testo, ma a scapito della sua leggibilità. È consigliabile invece risparmiare memoria ponendo più istruzioni su una stessa linea poichè i cinque byte del link, numero di linea e byte zero di fine linea, sono memorizzati una sola volta.

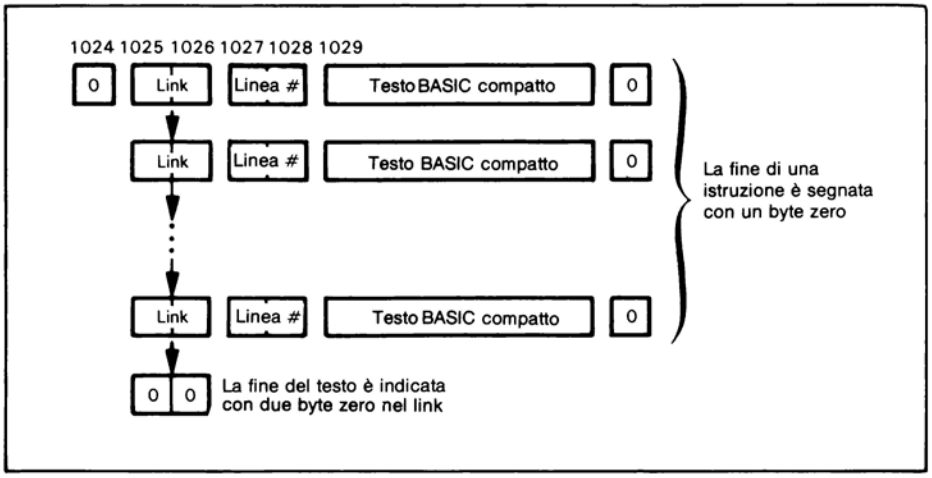


Figura 7-3: Memorizzazione delle istruzioni BASIC.

La lunghezza di ogni riga di istruzioni è variabile per cui viene posto un byte zero alla sua fine appunto per indicarne il termine. (Ricordiamo che la cifra zero sarebbe invece codificata con il byte 48). Il byte zero è riconosciuto dall'interprete BASIC nella fase di esecuzione del programma quando esso interpreta i vari simboli da sinistra verso destra. Il byte zero indica la file di una linea; i quattro byte successivi costituiscono il link e il numero di linea della istruzione seguente. Ricordiamo che il byte zero è impiegato per individuare la successione delle istruzioni mentre il link è usato per ricercare una istruzione a seconda del suo numero di linea. Tre byte zero consecutivi (quello di fine istruzione e i due di fine programma) danno la fine del testo di tutto il programma.

Un programma viene memorizzato su cassetta nello stesso formato come è presente in memoria centrale (Figura 7-3). Cioè a dire che esso è trasferito ("dumped") tale e quale sul nastro come un blocco continuo compresi i byte zero e i link.

L'uso di contrassegni, al posto delle parole-chiave, non è fatto solo dal BASIC CBM, ma anche da altri BASIC pur non esistendo un codice standard tra un interprete ed un altro. **Di conseguenza un programma BASIC salvato su un nastro**

CBM non è compatibile con altri sistemi ne viceversa altri (non CBM) programmi BASIC possono essere caricati ed eseguiti su un calcolatore CBM.

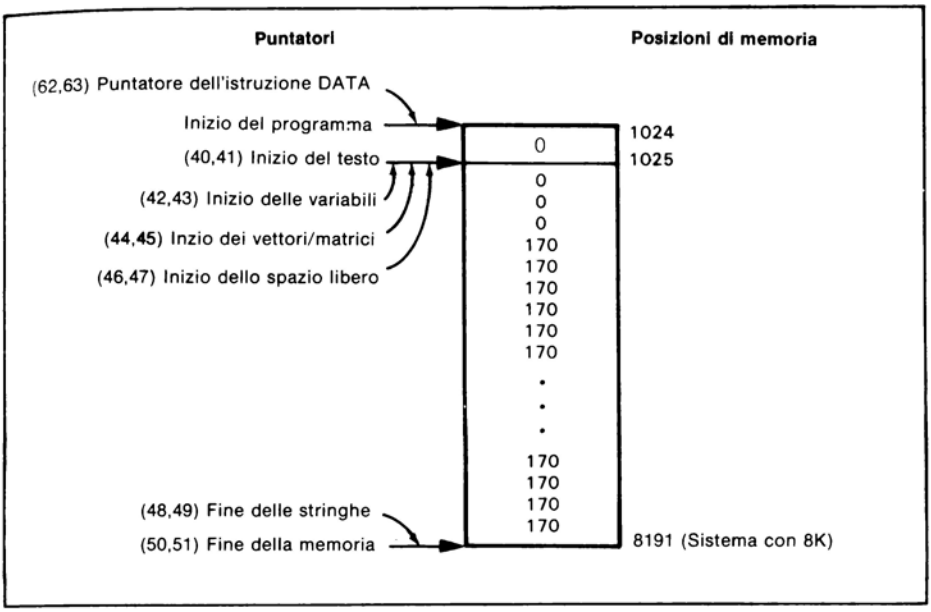


Figura 7-4: Area del programma di utente al momento dell'accensione.

INIZIALIZZAZIONE DELL'AREA PROGRAMMI DI UTENTE

All'accensione del calcolatore l'area programmi di utente è inizializzata con i caratteri "+" (codice 170) eccetto per le prime posizioni 1024, 1025 e 1026 in cui viene posto il valore zero. I puntatori sono inizializzati come illustrato nella Figura 7-4.

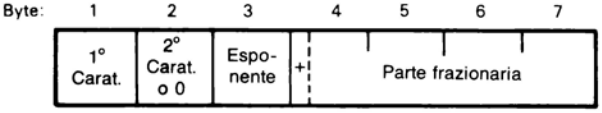
Man mano che le linee di programma entrano, o nuovi programmi sono caricati, il contenuto delle posizioni di memoria dell'area di utente cambia. Esso cambia tuttavia solo per quanto richiesto dal nuovo programma. L'area di utente non è continuamente ri-inizializzata (con "+" o qualunque altro codice), ma solo i puntatori sono ri-posizionati. **Sono infatti i puntatori che determinano l'estensione del programma corrente. La funzione del comando NEW per esempio è semplicemente quella di riportare i puntatori alla posizione iniziale come indicato nella Figura 7-4.** Nel caso invece del comando CLR i puntatori delle variabili e dei vettori sono portati alla fine del programma piuttosto che all'inizio come nel caso di NEW. **Infatti se per errore avete dato il comando CLR potete "leggere" l'area di utente e ripristinare opportunamente i puntatori.**

FORMATO DEI DATI

Variabili

Le variabili sono memorizzate nella zona, a loro riservata dell'area per programmi di utente (vedi Figura 7-2). Esse sono senza indice a differenza dei vettori e delle matrici che sono memorizzati in un'altra zona della memoria. Le variabili possono essere numeri interi, numeri reali o stringhe e sono liberamente mescolate nella loro zona di memoria. **Ogni variabile, qualunque sia il tipo, occupa sette byte di memoria.** I primi due byte contengono sempre il nome della variabile mentre gli altri cinque dipendono dal tipo di variabile. **Le variabili sono caricate nella tabella delle variabili man mano che sono incontrate durante l'esecuzione del programma.** Se una variabile non è elencata nella tabella, il BASIC presume che abbia valore zero, se variabile numerica, o nullo se stringa.

Formato delle variabili reali



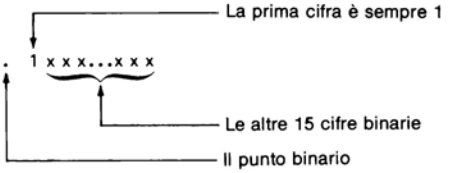
Il byte 1 contiene il primo carattere del nome della variabile. Il byte 2 contiene l'eventuale secondo carattere del nome e, se manca, contiene uno zero. I caratteri sono sempre codificati in ASCII (vedi Appendice A). Per esempio il nome A è codificato come 65,0 mentre il nome A0 come 65,48. **Le variabili reali sono caratterizzate dal fatto che il loro nome, in codice ASCII, non supera mai il numero 90.**

I byte dal 3 a 7 contengono il valore della variabile reale espresso in forma normalizzata standard con eccesso 128 (è la rappresentazione numerica nella forma mantissa-caratteristica).

Il byte 3 in particolare contiene l'esponente con eccesso 128. Con parole più semplici si può dire che l'esponente determina la grandezza del numero. Il formato con eccesso 128 significa che si è aggiunto 128 al valore vero dell'esponente dopo che il numero è stato normalizzato; di conseguenza l'esponente più piccolo contiene tutti zeri mentre quello più grande contiene tutti uno. Per esempio se il vero esponente è zero la sua rappresentazione è 128 (0 + 128). In pratica l'uso dell'eccesso elimina la necessità di considerare il segno dell'esponente. Ecco alcuni esempi:

Esponente attuale	Esponente memorizzato	Valore approssimato	
127	255	10^{38}	(esponente massimo)
34	162	10^{10}	
−1	127	10^{-1}	
−126	2	10^{-38}	
−128	0	10^{-39}	(esponente minimo, il numero è zero)

I byte da 4 a 7 contengono le cifre significative del numero (mantissa). Un numero si dice normalizzato quando la prima cifra significativa (non nulla) è immediatamente a destra del punto decimale (ricordiamo che è consuetudine anglosassone usare il punto anzichè la virgola nella numerazione frazionaria). In altre parole la rappresentazione del numero, codificato in binario, ha la forma:.



Per risparmiare memoria il punto non viene memorizzato come anche il bit 1 più significativo che deve ovviamente essere sempre pari a 1. La posizione del bit 1 viene utilizzata per memorizzare il segno: 0 per i numeri positivi e 1 per quelli negativi. Per normalizzare un numero, cioè portarlo nella forma vista sopra, si muove il punto verso sinistra e si incrementa l'esponente (per numeri grandi), o si muove il punto verso destra e si decrementa l'esponente (per numeri piccoli). Il numero zero è generalmente rappresentato con la cifra zero in tutti i byte da 3 a 7. È possibile che siano presenti dei residui di arrotondamento nella parte frazionaria, ma è sufficiente che l'esponente sia zero per dare a tutto il numero il valore zero.

Diamo alcuni esempi di rappresentazione di numeri reali memorizzati nell'area delle variabili. 1E + 38 ha l'esponente massimo di 255 che decresce verso zero mentre il numero tende a zero. Numeri frazionari come .5, .01 e .006 hanno l'esponente inferiore a 129. Nel caso di numeri negativi l'esponente aumenta da 0 a 255 con l'aumentare del valore assoluto del numero. Nel byte 4 il bit più significativo è riservato per il segno. In questa colonna i numeri inferiori a 127 hanno il bit 7 = 0 (numeri positivi) e i numeri maggiori di 127 hanno il bit 7 = 1 (numeri negativi).

Byte:	3	4	5	6	7
Numero	Esponente	±MSB		Frazione	LSB
1E+38	255	22	118	153	83
1E+10	162	21	2	249	0
1000	138	122	0	0	0
1	129	0	0	00	
0.01	122	35	215	10	62
1E−4	115	81	183	23	90
1E	62	60	229	8	101
1E−39	0	32	0	0	0
0	0	0	0	0	0
−1	129	128	0	0	0
−1000	138	250	0	0	0
−1E+10	162	149	2	249	0
−1E+38	255	150	118	153	83

Il seguente programma permette di esaminare la rappresentazione di numeri reali. Alla linea 10 si introduce un numero dalla tastiera. Alla linea 20 si punta all'inizio

della zona delle variabili; e poi si incrementa di 2 per andare oltre i due byte del nome della variabile A. Alla linea 30 si stampa il numero introdotto A seguito dai cinque byte estratti con il comando PEEK dalla tabella delle variabili. Il programma procede in maniera continua; per fermarlo premete RETURN.

```
10 INPUT A
20 X=PEEK(43)*256+PEEK(42)+2
30 PRINT A;"="PEEK(X);PEEK(X+1);PEEK(X+2);PEEK(X+3)PEEK(X+4)
40 GOTO 10
```

Formato delle variabili intere

Byte:	1	2	3	4	5	6	7
	1° carat. + 128	2° carat. + 128 o 128	Valore Alto	Valore Basso	0	0	0

Il byte 1 contiene il primo carattere del nome della variabile incrementato di 128. Il byte 2 contiene l'eventuale secondo carattere incrementato anche lui di 128. Se il secondo carattere manca, il byte 2 contiene esattamente il valore 128. Le variabili intere sono quindi caratterizzate dall'avere il loro nome, codificato in ASCII, con valore superiore o eguale a 176. Il simbolo di % non viene memorizzato. Nei byte 3 e 4 si memorizza il valore del numero intero nell'ordine byte alto e byte basso. (Vi ricordiamo che questo valore non è un indirizzo e non è quindi conforme all'inversione dei puntatori). **Un numero intero è memorizzato nella forma di complemento a 2.** Così il bit di ordine più significativo (bit 7 del byte 3) rappresenta il segno: 0 per i positivi e 1 per i negativi. **I rimanenti tre byte non sono usati e sono posti eguali a zero.**

Diamo qui di seguito alcuni esempi di rappresentazione di interi memorizzati nell'area delle variabili. Per leggere direttamente in memoria potete usare lo stesso programma di prima salvo cambiare A in A% nelle righe 10 e 30.

Byte	3	4
32767	127	255 (256•127+255=32767)
32766	127	254
14000	54	176
256	1	0
255	0	255
1	0	1
−1	255	255 (FFF_{16})+1=1
−2	255	254
−32766	128	2
−32767	1281	1

Formato delle variabili di stringa

Byte: 1	2	3	4	5	6	7
1° carat.	2° carat. + 128 o 128	Numero carat.	Puntatore Basso	Puntatore Alto	0	0

Il byte 1 contiene il primo carattere del nome della variabile di stringa. Il byte 2 contiene il secondo carattere incrementato di 128 o solamente 128 se manca il secondo carattere. Questa combinazione di valori ASCII determina appunto la presenza di una stringa. Anche in questo caso il simbolo $ non viene memorizzato. **Il byte 3 contiene la lunghezza in caratteri della stringa (1 − 255).**

La lunghezza della stringa è usata dalla funzione LEN. **I byte 4 e 5 contengono un puntatore all'inizio della stringa vera e propria che è memorizzata altrove.** Questo puntatore opera con lo standard previsto per il microprocessore 6502 con ordine byte basso/alto. I rimanenti due byte non sono usati e sono posti eguali a zero.

La memorizzazione delle stringhe è ottimizzata usando una sua copia se la stringa è già presente in memoria. Se una stringa è nuova essa viene creata e memorizzata nell'area delle stringhe nella parte superiore della memoria. Alcuni esempi saranno dati in seguito.

Costanti

Le costanti sono memorizzate nella stessa istruzione in cui sono citate. Cioè non sono poste in una zona separate della memoria ne tanto meno nell'area delle varibili. **Le costanti reali, intere e le stringhe sono tutte memorizzate in codice ASCII come già descritto per le istruzioni BASIC.** Per esempio la riga:

```
10 PRINT "HI!"
```

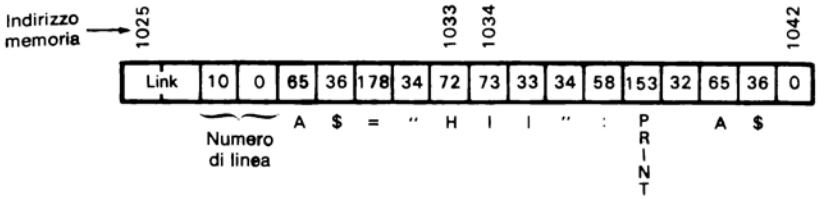
è memorizzata nell'area riservata alle istruzioni BASIC nella forma:

Link	10	0	153	32	34	72	73	33	34	0
	Numero di linea		P R I N T		"	H	I	I	"	

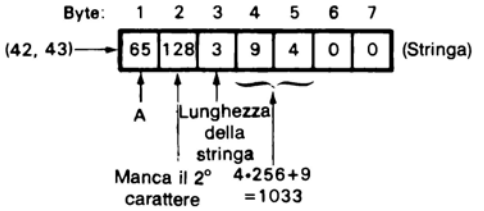
mentre invece l'istruzione:

```
10 A$="HI!":PRINT A$
```

è memorizzata in due aree distinte:

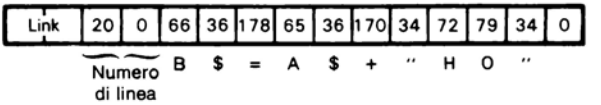


L'istruzione 10 vera e propria è memorizzata nell'area del BASIC a partire dall'indirizzo 1025 mentre invece la variabile A$, compare nell'area delle variabili, e fa riferimento al contenuto che è posto a partire dall'indirizzo 1033:

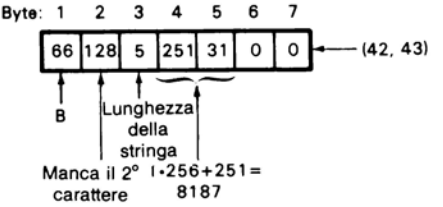


Notate infatti che la stringa A$, che viene citata due volte, fa sempre riferimento allo stesso contenuto che è posto a partire dalla posizione di memoria 1033. In un altro esempio invece una nuova stringa B$ ha il suo contenuto nell'area alta della memoria riservata appunto alle stringhe. Se scriviamo l'istruzione:

la codifica dell'istruzione sarà: `20 B$=A$+"HO"`

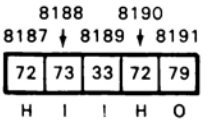


mentre quella della variabile B$ sarà:



In questo caso il puntatore punta ad un indirizzo della zona alta della memoria

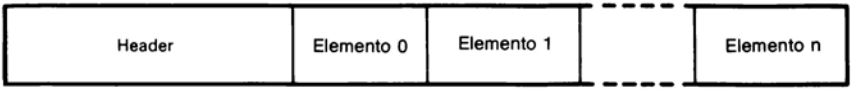
(8187) da cui inizia la stringa:



Ovviamente si è inteso che il calcolatore avesse 8K di memoria e quindi il massimo indirizzo è 8191.

FORMATO DELLA MEMORIZZAZIONE DELLE VARIABILI CON INDICI (VETTORI E MATRICI)

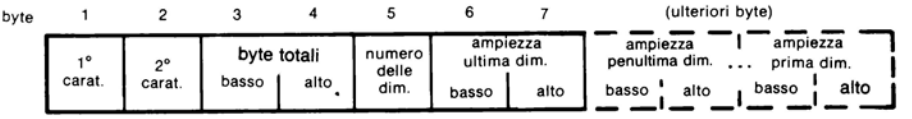
I vettori e le matrici sono memorizzati nell'area ad essi riservata nell'ambito della memoria di utente (vedi Figura 7-2). I vettori e le matrici possono essere costituiti da numeri reali, da numeri interi o da stringhe e sono memorizzati nell'ordine con cui sono creati dal programma. Il tipo di vettore o di matrice si distingue con lo stesso criterio con cui si distinguono le variabili senza indici e cioè con la diversa codifica dei caratteri che ne definiscono il nome. **Ogni vettore o matrice è memorizzato con una sua intestazione a cui seguono gli elementi che lo compongono:**



Nel caso di stringhe gli elementi sono memorizzati in ordine inverso.

Intestazione delle variabili con indici

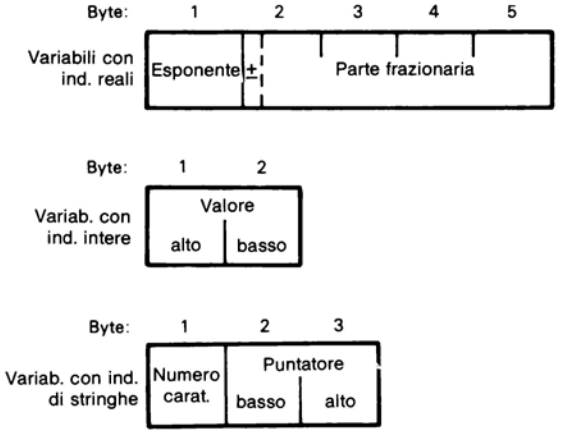
Il formato dell'intestazione "header" è sempre lo stesso qualunque sia il tipo di variabile. L'intestazione è composta da sette byte più due byte per ogni dimensione superiore alla prima.



Se poi gli elementi del vettore o della matrice sono reali essi sono memorizzati in cinque byte ognuno, con lo stesso formato che abbiamo già incontrato per le variabili reali. Se gli elementi sono invece numeri interi essi sono memorizzati in due byte ciascuno, mentre i puntatori delle stringhe sono memorizzati in tre byte ognuno.

I byte 1 e 2 dell'intestazione contengono il nome della variabile. I byte 3 e 4 contengono la lunghezza in byte totale di tutta la variabile, cioè la sua effettiva occupazione di memoria. Per esempio A(0) occupa 12 byte: 7 per l'intestazione e 5 per il suo unico elemento. Questa lunghezza è memorizzata con il solito criterio di byte basso/alto. **Il byte 5 contiene il numero degli indici (cioè se si tratta di un vettore a una dimensione o di una matrice a due o più dimensioni).** Così A(5) ha una sola dimensione (byte 5 = 1) e A(10, 10, 2) ha tre dimensioni (byte 5 = 3). **Nel caso di un vettore a una dimensione i byte 6 e 7 contengono l'ampiezza della dimensione** — cioè quel numero, che compare tra le parentesi dell'istruzione DIM — maggiorata di 1. Per esempio 61 per DIM A(60) e 101 per DIM B(100). Come ricorderete se un vettore non viene dimensionato con l'istruzione DIM, il Sistema gli attribuisce la dimensione fissa di 11. Anche l'ampiezza della dimensione viene memorizzata con lo standard di byte basso/alto. **Nel caso di variabili pluridimensionali l'intestazione contiene altri byte per dare l'ampiezza delle altre dimensioni. Per ogni ulteriore dimensione vengono aggiunti due byte. Le ampiezze dimensionali sono memorizzate in ordine inverso a quello che compare nella DIM.** Per esempio se abbiamo DIM A(10,5) i due valori dimensionali 11 e 6 sono memorizzati: nei byte 6,7 il 6 e nei byte 8,9 l'11. Se abbiamo invece DIM R(2,1,3) avremo 4 nei byte 6,7; 2 nei byte 8, 9 e 3 nei byte 10,11.

Qui di seguito riportiamo il formato per ogni tipo di elemento. Essi sono gli stessi già visti per le variabili senza indici salvo aver trascurato gli zeri.

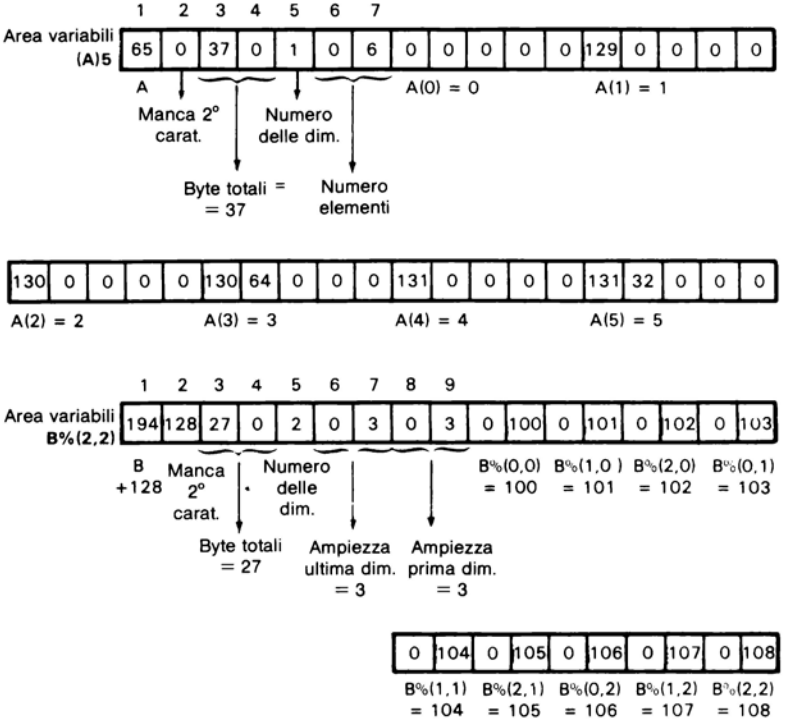


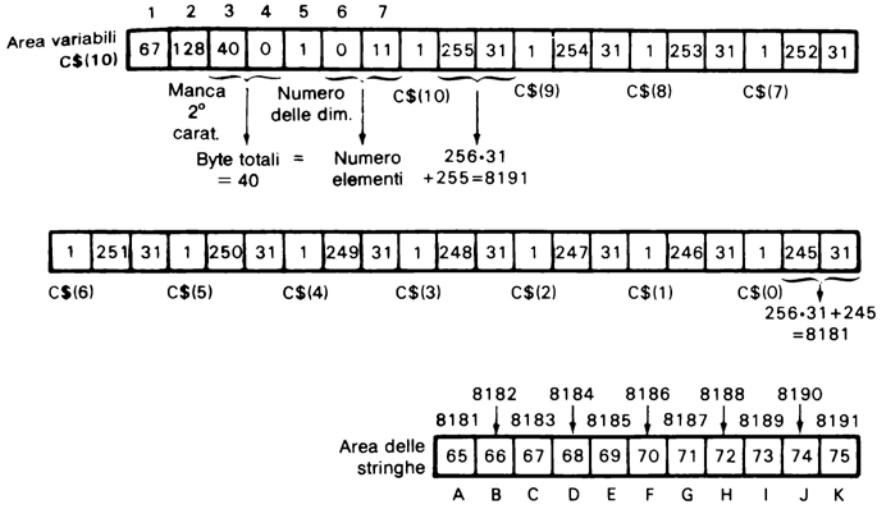
L'ampiezza in byte dell'intestazione può essere calcolata come pari a cinque più il doppio del numero delle dimensioni. La memoria occupata dai soli elementi si ottiene moltiplicando il numero dei byte per elemento (5 per i reali, 2 per gli interi e 3 per le stringhe) volte il numero degli elementi (le dimensioni più uno moltiplicate tra di loro). La lunghezza totale della variabile, pari alla somma dell'intestazione più lo spazio degli elementi, viene quindi memorizzata nel byte 4.

Il programma che qui di seguito vi proponiamo permette di esaminae il contenuto dell'area di memoria riservata alle variabili con indici:

```
10 DIM A(5),B%(2,2),C$(10):     REM ESEMPI DI ARRAY
20 FOR I=0 TO 5: A(I)=I: NEXT I
30 FOR I=0 TO 2: FOR J=0 TO 2: B(J,I)=100+3*I+J: NEXT J,I
40 FOR I=0 TO 10: C$(I)=CHR$(ASC("A")+I): NEXT I
50 X=PEEK(45)*256+PEEK(44):     REM PUNTAMENTO ALL'AREA DELLE ARRAY
60 Y=PEEK(47)*256+PEEK(46):     REM FINE DELLE ARRAY
70 FOR I=X TO Y
80 PRINT I, PEEK(I)
90 GET D$: IF D$="" GOTO 90
100 NEXT I
```

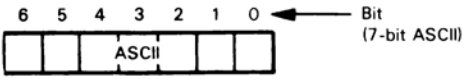
Alla linea 10 sono dimensionati i tre diversi tipi di variabili. Il vettore di numeri reali A viene caricato con i numeri da 0 a 5. La matrice B% viene caricata con i numeri interi da 100 a 108. Il vettore di stringhe C% infine è caricato con le lettere da A a K. Le linee 50 e 60 predispongono i puntatori alla fine dell'area delle variabili e alla fine dell'area dei vettori e matrici. La stampa si ferma ad ogni posizione di memoria esaminata, per proseguire battete un qualunque carattere (meglio se è RETURN).





RAPPRESENTAZIONE DEI CARATTERI

Il codice americano ASCII è un codice molto noto usato per la rappresentazione sia di caratteri alfanumerici che speciali. ASCII è l'acronimo di "American Standard Code fo Information Interchange". Esso è un codice normalmente a sette bit per cui permette la rappresentazione di 128 simboli diversi ($7F_{16} = 128_{10}$) che riportiamo nella Tabella A-2 dell'Appendice A. I bit sono numerati da 0 (bit meno significativo) a 6 (bit più significativo):



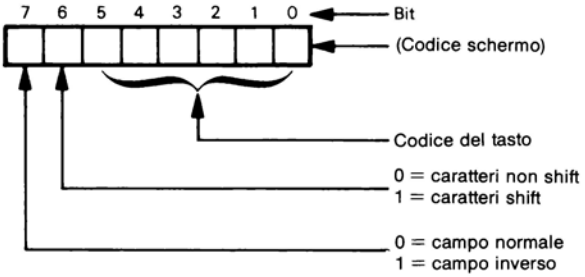
I primi 32 codici sono riservati per caratteri di controllo non stampabili quali i formati dei messaggi e i segnali di controllo della stampa.

I calcolatori CBM usano una versione estesa dell'ASCII che impiega otto bit. Grazie alla presenza di un ottavo bit è possibile codificare 256 caratteri. Per quanto riguarda invece i testi BASIC, in forma compatta, se l'ottavo bit è uguale a 1 si tratta di parole chiave mentre gli altri codici sono interpretati come in Tabella A-1. Altrove, nella memoria centrale, gli 8 bit sono decodificati come indicato nella Tabella A-4.

La memoria dello schermo, che occupa gli indirizzi da 32768 a 33767, usa invece una codifica ASCII, per i caratteri, diversa da quella usata nella memoria centrale. Come si vede nella Tabella A-3 questa memoria usa un codice a sette bit mentre l'ottavo bit è un indicatore per campi normali o inversi. Notate che i caratteri sono

organizzati in modo che i bit da 0 a 5 rappresentano i tasti della tastiera dei PET, mentre il bit 6 distingue i caratteri non-shifted (bit 6 = 0) da quelli shifted (bit 6 = 1).

L'insieme completo dei caratteri, usati nella memoria dello schermo, è riportato nella Tabella A-4 sotto la colonna PEEK/POKE.



Il codice ASCII della memoria dello schermo può essere derivato dal codice ASCII CBM muovendo il bit 7, del codice principale al 6° posto e trascurando il precedente valore di quest'ultimo bit. Nei quattro esempi che seguono si vede come lo 0 o 1 del bit 7 viene mosso nel bit 6:

Carattere	Rappresentazione in memoria centr.	Rappresentazione memoria schermo
	01000001	00000001
Shift A (♠)	11000001	01000001
1	00110001	00110001
Shift 1 (⊞)	10110001	01110001

Quando si esegue una istruzione di stampa PRINT sullo schermo il calcolatore CBM effettua automaticamente tale conversione. Solo quando si usano le istruzioni PEEK e POKE sulla memoria dello schermo si deve tener conto della diversa codifica.

La memoria dello schermo può essere vista come se avesse un ulteriore bit che seleziona i caratteri alternativi con POKE 59468,14. Per ripristinare i caratteri standard vi ricordiamo che dovete dare il comando POKE 59468,12. L'insieme di caratteri alternativo è riportato in Tabella A-4.

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLER

Il BASIC CBM può eseguire piccoli programmi scritti con il linguaggio Assembler del microprocessore 6502. Un programma scritto in Assembler ha sempre il vantaggio, rispetto ad un equivalente programma in BASIC, di richiedere meno occupa-

zione di memoria ed essere eseguito più velocemente. È probabile che desideriate scrivere in Assembler un vostro programma per uno dei seguenti motivi:

1. L'operazione scritta in BASIC è troppo lenta.
2. L'operazione non può essere scritta in BASIC.
3. L'operazione scritta in BASIC richiede troppo spazio in memoria.
4. L'Assembler è più adeguato al tipo di problema in oggetto come per esempio le routine di I/O.

Un programma in Assembler può essere caricato mediante comandi POKE che riportino il valore decimale delle istruzioni scritte nel linguaggio del microprocessore 6502. Non esiste una area di memoria riservata per i programmi in Assembler per cui dovete usare l'area per i programmi di utente o qualche zona non usata dal Sistema. Ecco alcune possibilità:

1. **Buffer delle unità a cassette.** Se non avete una seconda unità a cassette potete usare i 192 byte del suo buffer per depositarvi i vostri programmi in Assembler. Vi ricordiamo che questo buffer si trova tra gli indirizzi 826 e 1017 (vedi Appendice F). Inoltre se la prima unità a cassette non viene usata mentre girano programmi in Assembler potete usare allora anche il buffer 1 che si trova tra gli indirizzi 634 e 825. Ovviamente mentre gira il programma Assembler non potete effettuare operazioni LOAD, SAVE o altre che accedono alle cassette.
2. **Fine della memoria RAM.** Le posizioni 52 e 53 contengono un puntatore alla cima della memoria che nel caso dei PET a 8K ha l'indirizzo 8192. Si può quindi spostare temporaneamente il contenuto di questo puntatore ad un valore più basso così da riservare una parte della memoria, ora non più accessibile, ai vostri programmi Assembler. Se per esempio volete spostare in basso il puntatore di 1000 byte dovrete inserire il valore 7192 (8192-1000) convertito nei due indirizzi di ordine basso e alto:

$$7192_{10} = 1C18_{16} \rightarrow \overset{\text{alto}}{1C_{16}} = 28_{10} \text{ e } \overset{\text{basso}}{8_{16}} = 24_{10}$$

   Così 24 deve essere memorizzato nella posizione 52 (byte basso) e 28 nella posizione 53 (byte alto). Le istruzioni che potete dare in BASIC sono le seguenti:

```
10 AL=PEEK(52): AH=PEEK(53):    REM SALVA ATTUALE PUNTATORE
20 POKE 52,54: POKE 53,28:      REM CIMA DELLA MEMORIA 7192

100 POKE 52,AL: POKE 53,AH:     REM RIPRISTINA PUNTATORE
110 END
```

3. Anche **nell'area delle istruzioni BASIC** è possibile trovare della memoria libera. Potete infatti creare un blocco artificiale di dati ("dummy") mediante l'istruzione DATA. Ci sono poi alcune posizioni di memoria generalmente libere tra la fine del programma e l'inizio dell'area delle variabili, ma dovete fare moltissima

attenzione che il programma BASIC e il suo interprete non collidano con il programma Assembler.

Mediante l'interprete BASIC CBM è possibile caricare, un programma scritto in Assembler, con un metodo che è però alquanto rudimentale. Esso consiste nel tradurre dapprima in decimale le istruzioni numeriche del linguaggio del 6502 e poi caricarle nella memoria CBM mediante l'istruzione POKE. In altre parole, scrivete il vostro programma nel linguaggio Assembler del 6502 (vedere i testi consigliati nell'Appendice D) e poi dal codice esadecimale passate a quello decimale. Le singole istruzioni saranno poi caricate una per una con i comandi POKE.

Il monitor TIM della Commodore (Terminal Interface Monitor) memorizza invece direttamente i codici esadecimali, ma non permette di caricare contemporaneamente programmi in BASIC. Il vantaggio di caricare l'Assembler con i comandi POKE è quello di poter gestire tali programmi come routine di un programma principale BASIC. In conclusione mediante le istruzioni DATA/READ potete passare il testo Assembler al ciclo di caricamento POKE.

Esistono due possibilità per trasferire il controllo ad un programma Assembler: mediante le funzioni SYS o USR che sono in pratica equivalenti. SYS è una funzione che ha proprio lo scopo di trasferire il controllo ad un sottoprogramma indipendente. USR è una funzione di riferimento che permette di trasferire un parametro ad una routine in Assembler e di ritornarne un altro al programma principale.

Ovviamente la routine in Assembler deve ritornare il controllo, al programma principale in BASIC, mediante una istruzione di Ritorno da Subroutine RTS.

SYS

Sys è una funzione di sistema che trasferisce il controllo ad una routine indipendente. Il suo formato è:

```
SYS (indirizzo)
```

dove:

indirizzo	è una costante numerica, una variabile o una espressione che rappresenta l'indirizzo iniziale della routine. Tale valore deve essere compreso tra 0 e 65535.

A differenza di altre funzioni, SYS può essere data solo in modo differito. Per esempio:

`SYS(826)`	Trasferisce il controllo, in modo immediato, all'indirizzo di memoria 826 (secondo buffer per le cassette).
`55 SYS(826)`	Come il precedente, ma usato in modo differito. Al termine il programma BASIC prosegue con l'istruzione subito successiva a SYS.
`126 SYS(A+14`	Trasferisce il controllo all'indirizzo A + 14.

SYS, in altre parole, è l'equivalente di GOSUB, ma con la grande differenza che esclude le protezioni contro gli errori dell'utente previsti dal BASIC CBM. È molto

più frequente quindi che un programma Assembler si blocchi per errore, di quanto avvenga normalmente in BASIC durante le prime esecuzioni di prova.

Per ritornare al programma principale ricordatevi di inserire l'istruzione RTS nella vostra routine.

Valori e parametri possono essere passati tra il programma principale BASIC e la subroutine SYS mediante i comandi PEEK e POKE.

USR

USR è una funzione di sistema che passa un parametro ad una routine scritta in Assembler dall'utente, il cui indirizzo è contenuto nelle posizioni di memoria 1 e 2; successivamente ritorna un parametro al programma principale. Il suo formato è:

USR (dato numerico)

dove: dato numerico — è il parametro numerico passato alla subroutine.

Per esempio:

```
?USR(60)
```

Visualizza in modo immediato il valore ritornato dalla subroutine USR quando le si passa il valore 60.

```
105 A=USR(60)
```

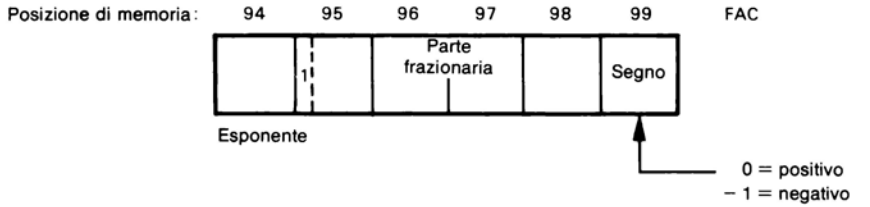
Come sopra, ma in modo differito.

```
210 IF USR(X)<4 GOTO 50

510 BM=USR(XA)+USR(3.4)+SQR(Y)+π
```

Desideriamo chiarire che prima di effettuare l'istruzione USR bisogna caricare l'indirizzo di inizio della subroutine nelle posizioni di memoria 1 e 2. Per esempio se la subroutine è posizionata nel secondo buffer delle unità a cassette, cioè all'indirizzo 826, voi dovete prima inserire le istruzioni:

		Alto	Basso
10 POKE 1,58			
20 POKE 2,3	$826_{10}=033A_{16}$	$3A_{16}=58_{10}$	$03_{16}=3_{10}$

Il valore del parametro è trasferito alla subroutine tramite alcune posizioni di memoria che funzionano come un accumulatore per numeri reali (FAC) per ogni possibile funzione. L'accumulatore FAC occupa 6 byte dall'indirizzo 94 al 99 ($5E_{16}$ a 63_{16}). Il formato del FAC è il seguente:



Come nel caso delle variabili reali, anche per questo accumulatore, l'esponente è memorizzato con eccesso 128 e la parte frazionaria è normalizzata con il bit di ordine più alto, del byte 95 (cioè il byte di ordine più alto), posto eguale a 1. La differenza tra questo formato e quello delle variabili è che il bit di ordine più alto, sempre eguale a 1, in questo caso viene tenuto presente (byte95). Un ulteriore byte (99) contiene il segno e ciò è fatto per rendere più facile la gestione dell'accumulatore.

La subroutine USR deve a sua volta andare a prendere il parametro nell'accumulatore FAC e portarvi alla fine l'eventuale nuovo parametro da trasferire al programma principale. Se la subroutine non altera il contenuto dell'accumulatore, tale valore ritornerà così come è al programma principale.

FILE AD ACCESSO DIRETTO

Gli archivi, o file, ad accesso diretto si possono creare indirizzando direttamente i blocchi sul dischetto e i buffer in memoria.

Ricordiamo che ogni blocco di dati occupa un settore del dischetto per cui è possibile creare un file, ad accesso diretto, indirizzando direttamente i settori e le tracce. I buffer di memoria, analogamente, sono anche loro direttamente indirizzabili e assegnati agli indirizzi secondari dei file logici. (Ricordiamo che ogni unità a dischetti ha 16 buffer di memoria di 256 byte ognuno).

I file ad accesso diretto sono creati mediante un certo numero di routine che accedono direttamente alla superficie del disco e ai buffer di memoria. Queste routine sono dello stesso tipo di quelle che implementano i file sequenziali e relativi, ma in questo caso siete voi che create la struttura operativa dei campi, dei record e dei file.

Vi consigliamo di non scrivere routine per file ad accesso diretto sino a che non sarete dei programmatori molto esperti. Dovrete infatti portarvi allo stesso livello dei progettisti che hanno scritto il software per file che già avete trovato nel BASIC CBM!

L'accesso diretto ai dischetti è programmato mediante istruzioni PRINT # che abbiano nella loro lista parametrica delle stringhe opportunamente codificate. Queste istruzioni PRINT # accedono al canale di comando tramite l'indirizzo secondario 15. I file logici ad accesso diretto vengono aperti con buffer di memoria assegnati ad ognuno di essi tramite i loro indirizzi secondari. La lista parametrica dell'istruzione PRINT # usa infatti l'indirizzo secondario per identificare i file logici e i buffer di memoria assegnati.

Il formato di una istruzione OPEN usata per aprire un file ad accesso diretto è il seguente:

```
OPEN lf, dev, sa, "# [bu]"
```

dove:	lf	è il numero di file logico
	dev	è il numero di periferica (normalmente 8)
	sa	è l'indirizzo secondario (compreso tra 2 e 14)
	bu	se presente, è il numero di buffer usato dall'indirizzo secondario. Esistono 16 buffer di 256 byte; i primi tre sono usati dal DOS. I rimanenti sono disponibili, ma se non viene specificato "bu" viene assegnato il primo buffer disponibile.

Per determinare il numero del file assegnato potete usare una istruzione GET # immediatamente dopo aver aperto il file ad accesso diretto. In ogni caso l'istruzione GET # deve essere eseguita prima che ogni altra istruzione di ingresso o uscita acceda al file. Ecco un esempio di programma:

```
5 REM ASSEGNA IL BUFFER 5 ALL'INDIRIZZO SECONDARIO 4,
6 REM USATO DAL FILE LOGICO 2
10 OPEN 2,8,4,"#5"
20 PRINT DS$:  REM CONTROLLO STATO I/O
30 GET#2,A$:PRINT ASC(A$):REM VISUAL. NUMERO BUFFER PER CONTROLLO
40 PRINT DS$:  REM CONTROLLA ANCORA STATO I/O
50 CLOSE 2
60 STOP
```

Tramite una istruzione PRINT #, con il seguente formato, si trasmettono i comandi di accesso al file:

```
10 OPEN lf,8,15
20 PRINT # lf, "parametro"
```

In questo formato "parametro" identifica le operazioni di accesso al file. "Parametro" ha due parti: un comando e una lista parametrica. Il comando può essere posto in forma lunga che deve terminare con una virgola oppure in forma corta e in tal caso si assume che la lista parametrica inizi al quarto carattere di tutta la stringa. I parametri possono essere separati da virgole o da spazi o caratteri di salto. Ecco le abbreviazioni usate per indicare i parametri:

sa	L'indirizzo secondario indicato nell'istruzione OPEN del file logico dei dati
dr	Il numero di drive (0 o 1)
t	Il numero di traccia sul dischetto
s	Il numero di settore sulla traccia selezionata
p	Il puntatore di carattere nel buffer con valore da 0 a 255
adl	Il byte di ordine basso dell'indirizzo di memoria
adh	Il byte di ordine alto dell'indirizzo di memoria
ns	Numero di caratteri compreso tra 1 e 34
data	Una stringa di dati di nc caratteri

adl, adh e nc devono essere indicati come argomenti della funzione CHR$. Per esempio se adl avesse il valore 123, dovremmo porre CHR$(123).

Lettura del blocco

Questa istruzione permette la lettura di un settore del dischetto e cioè ne trasferisce il contenuto nel buffer. Il formato di "block read" è il seguente:

```
PRINT # lf, "BLOCK-READ:sa,dr,t,s"
PRINT # lf, "B-Rsa,dr,t,s"
```

Il seguente esempio apre il file logico 2, assegna il buffer 5 all'indirizzo seconda-

rio 4, quindi legge il settore 0 della traccia 18 sul drive 1 per trasferirlo nel buffer 5:

```
10 REM APRE IL FILE LOGICO 2
11 REM ASSEGNA IL BUFFER 5 ALL'INDIRIZZO SECONDARIO 4
20 OPEN 2,8,4,"#5"
30 REM LEGGE SETTORE 0 DELLA TRACCIA 18 NEL DRIVE 1 CON BUFFER 5
40 OPEN 15,8,15
50 PRINT#15,"B-R4,1,18,0"
60 REM VISUALIZ. CONTENUTO BUFFER A CONFERMA CHE LO HA TROVATO
70 REM VISUALIZ. 256 BYTE DEL BUFFER SU 8 RIGHE E 32 COLON.
75 PRINT"J";
80 FOR I=1 TO 8
90 FOR J=1 TO 32
100 GET#2,A$: IF A$="" THEN 100
110 PRINT ASC(A$);
120 NEXT J
130 PRINT
140 NEXT I
150 CLOSE 2
160 CLOSE 15
170 STOP
```

Scrittura del blocco

Con l'istruzione "block write" trasferite il contenuto di un buffer in un settore; questo è il suo formato:

```
  PRINT # lf,"BLOCK-WRITE:sa,dr,t,s"
o PRINT # lf,"B-Wsa,dr,t,s"
```

Le seguenti istuzioni aprono il file logico 2, assegnano il buffer 8 all'indirizzo secondario 7 e trasferiscono il contenuto del buffer 8 nel settore 10 della traccia 35 del drive 0:

```
200 OPEN 2,8,7,"#8"
210 OPEN 15,8,15
220 REM L'ISTRUZ. CHE SCRIVE NEL BUFFER 8 DEVE SEGUIRE QUI.
300 PRINT#15,"B-W7,0,35,0"
310 CLOSE 2
320 CLOSE 15
330 STOP
```

Esecuzione del blocco

"Block execute" equivale a "block read" salvo che si usa quando i dati letti dal settore sono il codice oggetto di un programma in Assembler. Appena caricato tale programma può essere subito eseguito purchè termini con l'istruzione "return from subroutine" (RTS). Il formato di "block execute" è il seguente:

```
  PRINT # lf,"BLOCK-EXECUTE:sa,dr,t,s"
o PRINT # lf,"B-Esa,dr,t,s"
```

Puntatore del buffer

L'istruzione "buffer pointer" muove il puntatore del buffer dall'inizio del buffer a qualunque altra sua posizione; esso ha questo formato:

```
  PRINT # lf,"BUFFER-POINTER:sa,p"
o PRINT # lf,"B-Psa,p"
```

Se aggiungete la seguente istruzione all'esempio citato per l'istruzione "block read" ottenete lo spostamento del puntatore del buffer alla posizione 24:

```
55 PRINT#15,"B-P4,26"
```

Assegnazione dei blocchi

L'istruzione "block allocate" aggiorna la mappa di disponibilità dei blocchi (BAM) per mostrare come sono utilizzati. Tale mappa viene scritta sul dischetto quando il file logico è chiuso. Se il blocco (o settore) richiesto è già stato assegnato, allora il canale di errore identifica il prossimo blocco libero e viene segnalato l'errore NO BLOCK. Se nessun blocco è libero vengono allora ritornati i valori 00 come parametri di traccia e settore. "Block allocate" ha il seguente formato:

```
  PRINT # lf,"BLOCK-ALLOCATE:dr,t,s"
o PRINT # lf,"B-Adr,t,s"
```

Scrittura della memoria

L'istruzione "memory write" scrive i dati nel buffer. Essa ha il seguente formato:

```
PRINT # lf,"M-W"adl/adh/nc/data
```

Tabella 7-2. Indirizzo di inizio dei buffer nella unità a dischetto CBM 2040 e 8050 (256 byte per buffer)

Buffer N°	Modello 2040/8050	
	Esadecimale	Decimale
0	1000	4096
1	1100	4352
2	1200	4608
3	1300	4864
4	2000	8192
5	2100	8448
6	2200	8704
7	2300	8960
8	3000	12288
9	3100	12455
10	3200	12800
11	3300	13056
12	4000	13312
13	4100	13568
14	4200	13824
15	4300	14080

Gli indirizzi per i buffer di memoria, nel caso dei drive modello 2040 e 8050, sono riportati nella Tabella 7-2. Notate che tali indirizzi si presentano alquanto sparpagliati!

Supponete di dover scrivere, nel buffer 2 del drive modello 2040, i byte 32, 0, 17 e 36. Dalla Tabella 7-2 troviamo che l'indirizzo di partenza di questo buffer è 1200_{16} ($1200_{16} = 4608_{10}$; ripartendo il 1200_{16} nel byte basso resta 00_{10} e il byte alto 12_{16} diviene 18_{10}). L'istruzione PRINT # che dobbiamo scrivere è quindi la seguente:

```
100 PRINT#15,"M-W"CHR$(00)CHR$(18)CHR$(32)CHR$(0)CHR$(17)CHR$(96)
```

Lettura della memoria

L'istruzione "memory read" permette la lettura di un byte dal buffer. Essa ha il seguente formato:

```
PRINT # lf,"M-R"adl/adh
```

L'indirizzo del byte che volete leggere è specificato nella lista parametrica mediante la funzione CHR$. Il byte stesso viene poi letto con GET # tramite il canale di controllo (15). Di conseguenza una istruzione INPUT # non sarà eseguita correttamente sino a che una istruzione di accesso diretto, diversa da "memory read", "memory write" e "memory execute" sia stata eseguita.

Nel seguente esempio viene letto un byte dall'indirizzo 1208_{16} del buffer:

```
100 PRINT#15,"M-R"CHR$(8)CHR$(18)
110 GET#15,A$
```

"Memory execute"

Questa istruzione pone in esecuzione una subroutine in linguaggio Assembler. Essa ha il seguente formato:

```
PRINT # lf,"M-E"adl/adh
```

adl e adh sono le due parti di ordine basso e alto dell'indirizzo di partenza della subroutine nel buffer di memoria del dischetto. La subroutine deve terminare con la seguente istruzione di "ritorno dalla subroutine":

```
RTS,$60
```

"User"

Esistono dieci speciali istruzioni "user". Le prime due sostituiscono "block read" e "block write". Sette sono subroutine di salto "jump to" mentre la decima

Tabella 7-3. Istruzioni "User" per file ad accesso diretto

Designazione User	Designazione alternativa	Funzione
U1	UA	equivale a BLOCK-READ
U2	UB	equivale a BLOCK-WRITE
U3	UC	salto a $1300
U4	UD	salto a $1303
U5	UE	salto a $1306
U6	UF	salto a $D008
U7	UG	salto a $D00B
U8	UH	salto a $D00E
U9	UI	salto a $D0D5
U:	UJ	accensione $E18E

richiama la routine di accensione. Queste istruzioni sono riportate nella Tabella 7-3. Per le istruzioni da U3 a U9 vi rimandiamo alla revisione 3 della mappa di memoria dell'Appendice F ove potrete avere la loro interpretazione.

Per U1 e U2 usate il seguente formato:

```
PRINT # If "Ux;sa,dr,t,s"
```

dove X è uguale a 1 per U1 e a 2 per U2.

CAPITOLO 8

BASIC CBM

In questo capitolo descriviamo la sintassi di tutte le istruzioni e le funzioni del BASIC CBM. Dapprima presentiamo in ordine alfabetico le istruzioni e poi le funzioni.

Nei capitoli precedenti abbiamo visto i concetti fondamentali della programmazione ed abbiamo quindi descritto molte istruzioni e funzioni. In questo capitolo vogliamo invece dare l'esatto formato di tutte le istruzioni e di tutte le funzioni.

Modo immediato e modo differito

La maggior parte delle istruzioni possono essere eseguite in ambedue i modi. Solo in casi eccezionali esse sono eseguibili in uno solo dei due modi e in tal caso vi verrà detto esplicitamente.

Revisioni del BASIC

Alcune istruzioni e alcune funzioni sono disponibili solo con il BASIC 4.0 mentre la maggior parte sono eseguibili con qualunque versione di BASIC.

Talvolta, quando è possibile, vengono indicate le equivalenze tra queste due famiglie di interpreti BASIC. Le istruzioni del BASIC 4.0 richiedono il DOS versione 2.0 o superiori.

CONVENZIONI SUL FORMATO

In questo capitolo useremo le seguenti convenzioni nella descrizione del formato delle singole istruzioni.

LETTERE MAIUSCOLE	Le lettere e le parole in maiuscolo devono apparire esattamente così come sono indicate.
lettere minuscole	Le lettere e le parole in minuscolo hanno un impiego non determinato; il loro esatto uso viene stabilito dal programmatore.
{ }	Le parentesi graffe indicano una possibilità di scelta; esse stesse non appaiono nelle istruzioni.
[ ]	Le parentesi quadre indicano che il parametro è opzionale; esse stesse non appaiono nell'istruzione.
	I puntini indicano che la voce precedente può essere ripetuta; essi stessi non appaiono nell'istruzione.
numero di linea	È obbligatorio numerare le linee di un programma.

Nelle definizioni di funzioni e istruzioni che daremo le parole che seguono sono intese con questo significato:

accesso	lo scopo per cui un file di dati viene richiamato. Usate WRITE per una operazione di scrittura e READ per una di lettura
bno	il numero di carattere in un record di un file relativo
byte	un valore numerico compreso tra 0 e 255
condizione	una relazione del tipo: var { < > = < > <= >= } [espressione] se l'espressione posta a destra manca, allora la condizione è come se fosse: var = 0
costante	un qualunque valore fisso numerico o di stringa
< CR >	un carattere di ritorno del carrello
c$	una stringa o una funzione CHR$ che rappresentino una virgola, un ritorno del carrello o un altro separatore valido nella lista parametrica di una PRINT #
d	un drive di dischetto destinazione (0 o 1)
dato	una costante, una variabile o una espressione
daton	una dato numerico
dato$	un dato di stringa
Dd	un numero di drive di dischetto destinazione che deve essere indicato come D0 o D1
dev	il numero di una unità fisica periferica (vedi Tabella 8-1)

Tabella 8-1: Numeri delle unità fisiche

Numero	Periferica
0	Tastiera
1 ("default")	Unità a cassette # 1
2	Unità a cassette # 2
3	Schermo video
4	Stampante
5 — 7	Unità collegate alla porta IEEE
8	Unità a disco
9 — 30	Unità collegate alla porta IEEE
31 — 255	Non usati

dr	il numero di un drive di dischetto (0 o 1)
Ds	un numero di drive di dischetto sorgente che deve essere indicato con D0 o D1
< ESC >	il carattere o il tasto "escape"
espressione	una espressione aritmetica contenente operatori, costanti numeriche e variabili
filedest	il nome di un file destinazione
filesorgente	il nome di un file sorgente
istruzione	una istruzione BASIC
Ivv	il numero di un dischetto compreso tra 00 e 99 e indicato così: da I00 a I99
If	un numero di file logico

Tabella 8-2: Indirizzi secondari

Periferica	Indirizzo secondario	Operazione
Unità a cassette CBM	0 ("default") 1 2	Apertura per lettura Apertura per scrittura Apertura per scrittura; alla chiusura viene posto anche un segno EOT (End Of Tape)
Stampanti CBM	0 ("default") 1 2 3 4 5 6 7* 8* 9* 10*	Stampa normale Stampa sotto il controllo di un formato Memorizzazione del formato Dichiarazione del numero di righe per pagina Abilitazione della stampa dei messaggi diagnostici Definizione di caratteri personali Variazione della spaziatura tra le righe Stampa in minuscolo Stampa in maiuscolo Spegne l'unità 4 Reset
Unità a disco CBM	0 1 2 — 14 15	Non definito Non definito Apertura in lettura e/o scittura Accesso al canale di comando

* Solo per le stampanti con le nuove ROM

linea	il numero di una linea di programma
lineai	una specifica linea di programma
Ly	lunghezza di un record di un file relativo dove y è il numero di caratteri del record compreso tra 1 e 254. La lunghezza del record deve essere indicata con il formato da L1 a L254.
memoind	un indirizzo di memoria (compreso tra 0 e 65536)
messaggio	un testo di stringa racchiuso tra virgolette
nomedisco	il nome assegnato ad un disco
nomefile	il nome di un file
nomenuovo	il nome nuovo di un file
nomevecchio	il nome vecchio di un file
nvar	il nome di una variabile numerica
ON Uz	in BASIC 4.0 ha il significato di indicare il numero z di una unità fisica. Se manca viene imposta l'unità fisica numero 8 (cioè l'unità a disco).
rno	il numero di record in un file di dati relativo
< RVS >	il tasto REVERSE non shiftato
s	un drive di dischetto sorgente (0 o 1)
sa	un indirizzo secondario (vedi Tabella 8-2)
tipo	il tipo di un file di dati: SEQ per file sequenziale, PRG per file di programma, USR per file ad accesso diretto e REL per un file relativo
var	una variabile numerica o di stringa
var(i)	una variabile numerica o di stringa con indice
vv	un numero di dischetto compreso tra 00 e 99
W	un parametro che indica in un file sequenziale l'apertura in scrittura

APPEND # (BASIC 4.0)

L'istruzione APPEND # apre un file sequenziale già creato su dischetto e permette di aggiungere altri dati in coda al file. (Vedere anche PRINT # e COPY).

Formato:

```
APPEND # lf, "nomefile"  [,Dd] [ON Uz]
```

L'istruzione APPEND # apre il file di dati sequenziale "nomefile" sul dischetto nel drive d e posiziona il puntatore oltre la fine attuale del file. Le istruzioni PRINT# successive, con riferimento al file logico lf, possono scrivere nuovi dati che risultano quindi aggiunti. Se il parametro d è assente viene imposto il drive 0.

Esempio:

`APPEND#1,"CALC"` `PRINT#1,A`	Apre il file sequenziale "CALC" e poi vi scrive in coda il contenuto della variabile A
`APPEND#3,"TALK",D1` `PRINT#3,"123"`	Apre il file sequenziale "TALK" e poi vi scrive in coda la stringa "123"

BACKUP (BASIC 4.0)

L'istruzione BACKUP duplica un intero dischetto. Il duplicato e l'originale hanno lo stesso nome, numero di identificazione, intestazione, directory e gli stessi file. (Vedere anche PRINT # e DUPLICATE).

Formato:

```
BACKUP Ds TO Dd [ON Uz]
```

Il dischetto nel drive s viene duplicato. Il duplicato è nel drive d. Tutta l'operazione richiede alcuni minuti.

Esempio:

`BACKUP D0 TO D1`	Duplica il contenuto del dischetto nel drive 0 su quello nel drive 1
`BACKUP D1 TO D0`	Duplica il contenuto del dischetto nel drive 1 su quello nel drive 0

Attenzione: tutti i file sul disco sorgente devono essere in precedenza regolarmente chiusi prima di poter essere duplicati!

CLOSE

L'istruzione CLOSE chiude un file logico. (Vedere anche DCLOSE).

Formato:

```
CLOSE lf
```

L'istruzione CLOSE chiude il file logico lf. Se il parametro lf manca, allora vengono chiusi tutti i file in BASIC < 3.0. In BASIC 4.0, se manca lf, viene dato un errore di sintassi.

Dopo aver svolto le operazioni di lettura/scrittura i file devono essere chiusi. L'operazione di chiusura di un file può essere fatta una sola volta. Per i dettagli dell'operazione di chiusura vi rimandiamo al capitolo 6.

Esempio:

```
CLOSE 1      Chiude il file logico 1
CLOSE 14     Chiude il file logico 14
```

CLR

L'istruzione CLR pone tutte le variabili numeriche a zero e assegna il valore nullo alle variabili di stringa. La zona di memoria riservata alle variabili con indici viene resa libera. Questa istruzione equivale a spegnere il calcolatore, poi riaccenderlo e ricaricare il programma. CLR chiude tutti i file in quel momento aperti.

Formato:

```
CLR
```

Dopo il comando CLR il programma può continuare la sua esecuzione purchè CLR non ne modifichi la logica.

Esempio:

```
100 CLR
```

CMD

L'istruzione CMD invia all'unità fisica stampante tutti i dati in uscita che andrebbero invece al display. Le uscite vanno alla stampante, invece che al display, sino a che non sia eseguita una istruzione PRINT # che specifichi lo stesso numero di file logico. Almeno una istruzione PRINT # deve essere eseguita dopo CMD.

Formato:

```
CMD lf
```

L'istruzione CMD assegna il canale della stampante al file logico lf. Dopo l'esecuzione di CMD, le istruzioni PRINT e LIST stampano i dati anzichè visualizzarli. Vi rimandiamo al capitolo 6 per la programmazione della stampante.

Esempio:

La seguente sequenza usa CMD per stampare il listato di un programma.

OPEN 5,4	Apre il file logico 5 e seleziona la stampante
CMD 5	Invia le uscite sulla stampante
LIST	Stampa il listato di un programma
PRINT#5	Stampa un ritorno del carrello e toglie il collegamento verso la stampante
CLOSE 5	Chiude il file logico 5

COLLECT (BASIC 4.0)

L'istruzione COLLECT ricrea la "mappa di disponibilità dei blocchi" (BAM) per tutti i file di un dischetto. I file non correttamente chiusi saranno chiuso o cancellati.

Formato:

COLLECT [Dd] [ON Uy]

L'operazione di "collect" viene fatta sul dischetto nel drive d. Se il parametro Dd manca il sistema impone il drive 0.

Esempio:

COLLECT	Fa il riordino del dischetto posto nel drive richiamato più di recente
COLLECT D0	Fa il riordino del dischetto posto nel drive 0
COLLECT D1	Fa il riordino del dischetto posto nel drive 1

CONCAT (BASIC 4.0)

L'istruzione CONCAT concatena due file di dati. (Vedere anche PRINT # e COPY).

Formato:

CONCAT [Ds,] "filesorgente" TO [Dd,] "filedest" [ON Uz]

Il file sorgente (drive Ds) viene posto in successione al file destinazione (drive Dd). Il file sorgente rimane anche nella versione originale, mentre il file destinazione risulta dall'unione dei due file. Se i numeri di drive s e d mancano, allora viene imposto il valore 0.

Attenzione: I file devono essere chiusi prima di essere concatenati!

Esempio:

`CONCAT "FIRST" TO "SECOND"`	Il file "FIRST" viene concatenato al file "SECOND". I due file sono sul dischetto nel drive 0
`CONCAT D1,"ABC" TO D0,"XYZ"`	Il file "ABC" (dischetto nel drive 1) viene concatenato al file "XYZ" (dischetto nel drive 0)

CONT

L'istruzione CONT, battuta alla tastiera in modo immediato rimette in esecuzione un programma dopo un BREAK.

Formato:

CONT

Una istruzione "break" è causata dalle istruzioni STOP o END. È possibile inoltre interrompere l'esecuzione di un programma premendo il tasto STOP. L'esecuzione del programma riprende al punto esatto in cui era stata fermata.

Se in risposta ad una istruzione INPUT battete RETURN, avrete una interruzione "break". Battendo allora CONT l'istruzione INPUT sarà rieseguita.

Esempio:

CONT

COPY (BASIC 4.0)

L'istruzione COPY ricopia un singolo file oppure tutti i file di un dischetto. (Vedere anche PRINT # e COPY).

Formato:

COPY [Ds,] ["filesorgente"] TO [Dd,] ["filedest"] [ON Uz]

Nel primo caso, cioè per ricopiare un singolo file, il file sorgente (drive Ds) viene ricopiato sul dischetto destinazione (drive Dd) e prende il nuovo nome "filedest". Nella lista parametrica di COPY i due nomi di file devono essere indicati; Ds e Dd possono invece mancare e in tal caso sono posti eguale a 0.

Nel caso invece si desideri copiare tutti i file di un dischetto, posto ad esempio nel primo drive, su un altro dischetto, posto quindi nel secondo drive, non si devono indicare i due nomi dei file nella lista parametrica di COPY. In questo caso è però obbligatorio indicare i due numeri di drive Ds e Dd, che devono essere differenti.

Se il nome di un file sorgente è già presente sul dischetto destinazione, l'operazione di ricopiatura si interrompe e viene visualizzato il messaggio FILE ALREADY EXISTS.

Copy non modifica alcun file già presente sul dischetto destinazione.

Attenzione: Un file deve essere chiuso prima di poter essere copiato!

Esempio:

`COPY D1 TO D0`	Copia tutti i file posti sul dischetto nel drive 1 su quello nel drive 0. (Solo per le versioni DOS 2.0 o superiori)
`COPY D1, "MAJOR" TO D1, "MINOR"`	Crea il file "MINOR" sul dischetto nel drive 1

DATA

L'istruzione DATA dichiara dei valori costanti che saranno assegnati alle variabili di una istruzione READ.

Formato:

DATA costante [, costante, costante, ..., costante]

L'istruzione DATA può essere posta ovunque in un programma e può indicare sia costanti numeriche che di stringa. Le costanti di stringa sono normalmente racchiuse tra virgolette. Le virgolette sono necessarie per le stringhe contenenti caratteri grafici, spazi vuoti, virgole e punti e virgole. Se la stringa non è racchiusa tra virgolette gli spazi vuoti, le virgole, i punti e virgole e i caratteri grafici sono trascurati. Un carattere virgolette non può essere indicato con la funzione CHR$(34).

L'istruzione DATA può essere usata solo in modo differito.

Esempio:

`10 DATA NAME,"C.D."`	Definisce due costanti di stringa
`50 DATA 1E6,-10,XYZ`	Definisce due costanti numeriche e una stringa

Per completare la descrizione dell'istruzione DATA vi rimandiamo all'istruzione READ.

DCLOSE (BASIC 4.0)

L'istruzione DCLOSE chiude uno o tutti i file aperti su dischetto. (Vedere anche CLOSE).

Formato:

DCLOSE # lf [ON Uz]

Se lf è specificato viene chiuso solo quel file; diversamente vengono chiusi tutti i file su dischetto ancora aperti.

Esempio:

`DCLOSE`	Chiude tutti i file aperti su dischetto
`DCLOSE#1`	Chiude il file numero 1 aperto su dischetto
`DCLOSE ON U8`	Chiude tutti i file aperti su dischetto dell'unità fisica 8

DEF FN

La funzione DEF FN permette di definire e usare funzioni speciali create dall'utente.

Formato:

DEF FNnvar (arg) = espressione

La variabile numerica nvar definisce univocamente la funzione che sarà denominata FNnvar ( ). nvar deve essere una variabile numerica frazionaria con al massimo cinque lettere. Se tale nome è più lungo, oppure viene posto il simbolo di variabile intera o di stringa, si avrà un errore di sintassi.

La funzione è definita da una espressione aritmetica composta da costanti numeriche, variabili, operatori. "arg" è una variabile fittizia ("dummy") che può anche apparire nell'espressione.

"arg" è l'unica variabile che può essere specificata solo al momento del richiamo di FNnvar (arg), mentre le altre eventuali variabili devono essere state già definite in precedenza.

Una intera istruzione DEF FN non può superare una linea di 80 caratteri. Tuttavia nella parte espressione possono essere presenti altre funzioni, create in precedenza, così che si può avere una funzione anche molto complessa e lunga.

Una funzione può essere ridefinita in un programma; cioè si può usare lo stesso nome nvar in un'altra DEF FN.

DEF FN non può essere usata in modo immediato. Se però DEF FN fa parte di un programma presente in memoria, allora FNvar può essere richiamata in modo immediato.

Esempio:

`10 DEF FNC(R)=π*R↑2`	Definisce una funzione che calcola la lunghezza della circonferenza. Essa riceve un singolo argomento R e ritorna il valore della circonferenza.
`?FNC(1)`	Stampa il valore 3.14159265 (π)
`A=FNC(14)`	Assegna alla variabile A il valore della funzione FNC calcolata con l'argomento 14
`55 IF FNC(X)>60 GOTO 150`	Usa il valore calcolato dalla funzione FNC in una istruzione di salto condizionato. La funzione FNC usa il valore corrente di X.

DIM

L'istruzione DIM riserva spazio in memoria per le variabili con indici.

Formato:

```
DIM var(i) [, var(i), var(i), ..., var(i)]
```

Le variabili possono avere uno o più indici:

var(i)	variabile a una dimensione vettoriale
var(i, j)	variabile a due dimensioni matriciale
var(i, j, K)	variabile a tre dimensioni matriciale

Per una descrizione dettagliata delle variabili con indici vi rimandiamo al capitolo 4.

Le variabili con indici devono essere sempre dimensionate, con l'istruzione DIM, prima di essere usate tranne il caso di variabili vettoriali con undici elementi. Queste ultime vengono automaticamente dimensionate quando sono richiamate per la prima volta in un programma.

L'operazione di dimensionamento deve avvenire una sola volta. In caso di errore il calcolatore vi avvisa con il messaggio ?REDIM'ED ARRAY e il programma viene fermato.

L'istruzione CLR permette di ripetere l'operazione di dimensionamento.

Esempio:

`10 DIM A(3)`	Dimensiona una variabile di 4 elementi con un indice
`45 DIM X$(44,2)`	Dimensiona una variabile con due indici e 135 elementi (45 per 3)
`1000 DIM MU(X,3*B),N(12)`	Dimensiona una variabile MU con due indici; X e B devono essere definiti prima di eseguire la DIM. Dimensiona anche la variabile N con un solo indice

DIRECTORY (BASIC 4.0)

L'istruzione DIRECTORY visualizza le "directory" di uno o ambedue i dischetti presenti nei drive. La parola CATALOG può essere usata in sostituzione di DIRECTORY. (Vedere anche LOAD "$dr").

Formato:

```
DIRECTORY [Dd] [ON Uz]
```

Viene visualizzata la directory del dischetto nel drive d. Se il parametro Dd manca vengono allora visualizzate ambedue le directory dei due dischetti.

Se il drive selezionato non contiene alcun dischetto, viene dato un errore di stato.

L'istruzione DIRECTORY viene normalmente eseguita in modo immediato.

Esempio:

`DIRECTORY`	Visualizza le directory dei dischetti nei drive 1 e 0
`DIRECTORY D1`	Visualizza la directory del dischetto nel drive 1

Stampa di una directory

Una directory può essere stampata, invece di essere visualizzata, aprendo il canale verso la stampante prima di eseguire l'istruzione DIRECTORY. Le istruzioni che potete dare sono le seguenti:

`OPEN 4,4`	Apre la stampante specificando il file logico 4
`CMD 4`	Commuta l'uscita verso la stampante
`DIRECTORY`	Stampa le due directory (drive 1 e 0) sulla stampante
`PRINT#4` `CLOSE 4`	Ri-commuta l'uscita verso il display

DLOAD (BASIC 4.0)

L'istruzione DLOAD carica un programma BASIC da un dischetto nella memoria centrale. (Vedere anche LOAD).

Formato:

DLOAD "nomefile" [Dd] [ON Uz]

L'istruzione DLOAD carica il programma "nomefile" dal dischetto nel drive Dd nella memoria centrale. Se Dd manca viene imposto il valore 0.

Esempio:

`DLOAD "CALC"`	Carica il programma CALC dal drive 0
`DLOAD "TIME",D1`	Carica il programma TIME dal drive 1
`A$="PROG"` `DLOAD A$`	Carica il programma PROG dal drive 0
`DLOAD"PROG",D0 ON U8`	Carica il programma PROG dal drive 0 dell'unità 8

In BASIC 4.0 premendo il tasto RUN/STOP, assieme allo shift, viene caricato automaticamente il primo programma incontrato sul dischetto e subito dopo viene posto in esecuzione.

DOPEN (BASIC 4.0)

L'istruzione DOPEN apre un file di dati per una operazione di lettura e/o scrittura.

Formato:

DOPEN # If, "nomefile" [,Ly] [,Dd] [ON Uz] [,W]

L'istruzione DOPEN apre il file di dati denominato "nomefile" sul dischetto nel drive Dd e gli assegna il numero di file logico If. Se Dd manca viene imposto il valore 0. Se manca Ly il file è sequenziale; in tal caso il parametro W indica l'apertura in scrittura e la sua mancanza quella in lettura.

Se Ly è presente il file è relativo con lunghezza di y byte per record. I file relativi sono aperti contemporaneamente sia per la lettura che per la scrittura per cui il parametro W non deve essere posto.

Esempio:

`DOPEN#1,"PRIZES"`	Apre il file sequenziale PRIZES nel drive 0 per operazioni di lettura
`DOPEN#6,"SNAKE"L30,D1`	Apre il file relativo SNAKE nel drive 1. I record hanno lunghezza di 30 caratteri e possono essere sia letti che scritti

DSAVE (BASIC 4.0)

L'istruzione DSAVE trascrive un programma BASIC dalla memoria centrale su un dischetto. (Vedere anche SAVE).

Formato:

DSAVE "nomefile" [,Dd] [ON Uz]

L'istruzione DSAVE "salva" un programma, presente nella memoria centrale, trascrivendolo come file denominato "nomefile" sul dischetto nel drive Dd. Se Dd manca viene posto il drive 0.

Esempio:

`DSAVE"TRUE"`	Scrive il file di programma TRUE sul dischetto nel drive 0
`DSAVE"FALSE",D1`	Scrive il file di programma FALSE sul dischetto nel drive 1

END

L'istruzione END pone termine all'esecuzione del programma e riporta il calcolatore al modo immediato.

Formato:

END

L'istruzione END permette di terminare un programma in punti diversi dalla sua ultima istruzione. Mettere una END dopo l'ultima istruzione non è obbligatorio.

END permette di separare più programmi presenti contemporaneamente in memoria.

L'istruzione END è usata solo in modo differito.

Esempio:

```
20001 END
```

FOR, NEXT E STEP

Tutte le istruzioni tra FOR e NEXT vengono eseguite uno stesso numero di volte.

Formato:

```
FOR nvar = iniz TO fine STOP incr
istruzioni del ciclo
NEXT [nvar]
```

dove:

nvar	è la variabile indice del ciclo che può essere anche usata nelle istruzioni interne.
iniz	è una variabile o costante numerica, oppure una espressione che stabilisce il valore iniziale dell'indice nvar.
fine	è una variabile o costante numerica, oppure una espressione che indica il valore finale dell'indice. Durante l'ultima esecuzione del ciclo l'indice assume un valore eguale o il più prossimo a fine.
incr	è una variabile o costante numerica, oppure una espressione. Ad ogni esecuzione del ciclo l'indice viene incrementato del valore incr. incr può essere positivo o negativo. Se manca viene posto il valore 1.

Non è obbligatorio citare l'indice nell'istruzione NEXT. Se più cicli sono nidificati e terminano nello stesso punto è possibile usare una sola NEXT:

NEXT $nvar_1$, $nvar_2$ ...

Un ciclo FOR NEXT viene sempre eseguito almeno una volta anche se il valore iniziale dell'indice è oltre il valore finale. Se NEXT manca il ciclo viene eseguito una sola volta.

I valori iniz, fine e incr sono letti una sola volta durante la prima esecuzione del ciclo e non possono più essere modificati. È possibile cambiare nvar e per esempio porla eguale al valore fine, per far terminare in anticipo il ciclo. È sconsigliabile uscire da un ciclo direttamente con GOTO.

Più cicli FOR NEXT possono essere posti a fascio o nidificati, ma un ciclo interno deve sempre essere completamente contenuto in quello esterno. Al massimo questi cicli possono avere l'istruzione NEXT in comune.

Esempio:

```
10 FOR IN = 0 TO 100
.
40 NEXT IN
100 FOR X = A + 14 TO C-64+D/2 STEP 4
.
150 NEXT X
60 FOR A1 = 50 TO 0 STEP -1
.
90 NEXT
100 FOR I = 0 TO 10 STEP 0.5
.
155 NEXT
250 FOR I = 1 TO 5
260 FOR J = A TO B
.
300 NEXT I,J    come    300 NEXT I    come    300 NEXT
                        310 NEXT J            310 NEXT
```

GET

L'istruzione GET permette di acquisire un singolo carattere alla volta dalla tastiera.

Formato:

```
GET var
```

L'istruzione GET può essere eseguita solo in modo differito. Al momento dell'esecuzione dell'istruzione GET, viene assegnato inizialmente a var il valore zero, se essa è numerica, e nullo se è una stringa. Subito dopo viene assegnato a var il carattere successivo che è presente nel buffer della tastiera. Attenzione quindi che se il buffer è vuoto, nel caso di var numerica, rimane il valore zero.

GET è usata per acquisire un carattere alla volta dalla tastiera e accetta anche il carattere RETURN che assegna a var come CHR$(13).

Se var è una variabile numerica e non viene battuto alcun tasto alla tastiera, le viene assegnato il valore zero.

Se var è numerica e viene battuto un tasto non numerico, il programma è interrotto e viene dato il messaggio ?SYNTAX ERROR.

L'istruzione GET può avere più parametri nella sua lista, ma questa possibilità è estremamente difficile da usare:

```
GET var, var, var
```

Esempio:

```
10 GET C$

10 GET D

10 GET A,B,C
```

GET

L'istruzione GET # Esterna (GET #) riceve un singolo carattere in ingresso da una periferica esterna identificata da un numero di file logico.

Formato:

```
GET # If, var
```

L'istruzione GET # può essere usata solo in modo differito. GET # assegna un carattere alla variabile var. La periferica esterna deve essere stata aperta in precedenza con OPEN o DOPEN.

GET e GET # gestiscono le variabili e i dati in ingresso nello stesso modo. (Vedere GET).

Esempio:

```
10 GET#4,C$:IF C$="" GOTO 10
```

Chiede un carattere dalla tastiera. Ri-esecuzione di attesa

GOSUB

L'istruzione GOSUB esegue il salto del programma ad una subroutine specificata con il numero di linea e permette il ritorno all'istruzione immediatamente successiva a GOSUB.

Formato:

```
GOSUB linea
```

Il numero di linea indicato in GOSUB è il primo numero di linea della subroutine inteso in senso logico. Esso può non essere il numero più piccolo della subroutine, ma è quello della prima istruzione che deve essere eseguita.

L'ultima istruzione logica della subroutine deve essere RETURN che permette il ritorno all'istruzione subito successiva a GOSUB nel programma principale.

Il salto GOSUB può essere posto in un programma. Più subroutine possono essere nidificate. Si possono avere sino a 26 salti a subroutine contemporanei. Questo significa che si possono eseguire sino a 25 istruzioni GOSUB prima di avere il primo RETURN.

Esempio:

```
100 GOSUB 2000      Salta alla subroutine della linea 2000
110 A = 3+2         Punto di ritorno dalla subroutine
.
.
2000                Punto di inizio della subroutine
.
.
2090 RETURN         Ritorno al programma principale
```

GOTO

L'istruzione GOTO esegue un salto incondizionato.

Formato:

```
GOTO linea
```

Il programma prosegue la sua esecuzione alla linea indicata come parametro "linea".

Esempio:

```
10 GOTO 100
```

GOTO può essere eseguita in modo immediato solo con riferimento ad un programma già presente in memoria. GOTO non può saltare ad istruzioni che siano prive del numero di linea.

HEADER (BASIC 4.0)

L'istruzione HEADER prepara un dischetto assegnandogli un nome e un numero di identificazione. (Vedere anche PRINT # PREPARE).

Formato:

```
HEADER "nomedisco", Dd [,Ivv] [ON Uz]
```

L'istruzione HEADER provvede a segnare i settori sulle tracce e a inizializzare la directory e la mappa di disponibilità dei blocchi BAM. Il dischetto assumerà il nome "nomedisco" e il numero di identificazione vv. Questo nome e questo numero saranno sempre visualizzati all'inizio della directory.

L'istruzione HEADER viene normalmente eseguita in modo immediato.

HEADER può essere usata sia con un dischetto nuovo che con un dischetto già usato. Quando viene usata in modo immediato, il calcolatore chiede una conferma ARE YOU SURE? a cui dovete rispondere YES (CR) per eseguire effettivamente la HEADER.

Se il dischetto è rovinato oppure se manca dal drive oppure se la tacca di protezione da scrittura è coperta, allora viene dato il messaggio ?BAD DISK.

Esempio:

```
HEADER "MASTER",D0,I02
```

Viene preparato un dischetto nel drive 0.
Il dischetto prende il nome "MASTER" e il numero 02.

IF ... THEN ...

L'istruzione IF THEN permette l'esecuzione di alcune istruzioni al verificarsi di una condizione.

Formato:

```
IF condizione THEN istruzione [: istruzione ...]
```

oppure anche:

```
IF condizione { THEN } linea
              { GOTO }
```

Se la condizione si verifica allora vengono eseguite le istruzioni o il salto dopo THEN. Se la condizione è falsa, allora viene eseguita l'istruzione posta sulla linea di programma subito successiva. In tal caso le istruzioni dopo THEN vengono ignorate.

Il salto condizionato può essere espresso nella forma composta THEN GOTO, come anche nelle altre forme:

```
IF A=1 THEN 50
IF A=1 GOTO 50      } equivalenti
IF A=1 THEN GOTO 50
```

Ricordiamo che tra le istruzioni che si possono porre dopo THEN vi può essere anche un salto incondizionato GOTO. Ovviamente dopo questo GOTO è inutile porre alcuna altra istruzione.

Le seguenti istruzioni non possono apparire in una istruzione IF THEN eseguita in modo immediato: DATA, GET, GET #, INPUT, INPUT #, REM, RETURN, END, STOP e WAIT.

Le istruzioni CONT e DATA non possono apparire in una IF THEN eseguita in modo differito.

Un ciclo FOR NEXT può seguire alla parola THEN purchè sia contenuto tutto sulla stessa linea. Anche un'altra IF THEN può seguire alla parola THEN purchè sia contenuta tutta sulla stessa linea. È preferibile però usare gli operatori Booleani per unire più IF THEN. Per esempio le due istruzioni seguenti sono equivalenti, ma è preferibile usare la seconda:

```
10 IF A$ = "X" THEN IF B = 2 THEN IF C > D THEN 50
10 IF A$ = "X" AND B = 2 AND C > D THEN 50
```

Esempio:

```
400 IF X > Y THEN A = 1
500 IF M+1 THEN AG = 4.5:GOSUB 1000
```

INPUT

L'istruzione INPUT permette di ricevere dati dalla tastiera.

Formato:

INPUT { (spazio) / "messaggio"; } var [,var, ..., var]

INPUT può essere usata solo in modo differito. Al momento dell'esecuzione di una INPUT appare sullo schermo del display un punto interrogativo "?". L'utente deve allora fornire i dati rispettando esattamente l'ordine e il tipo delle variabili poste nella lista di INPUT. Nel caso di più variabili i dati corrispondenti devono essere battuti con una virgola di separazione. L'ultimo dato deve essere seguito dal ritorno del carrello:

?1234 < CR >	ingresso di un solo dato
?1234,567, CIAO < CR >	ingresso di più dati

Se è presente un "messaggio" esso viene visualizzato prima del punto interrogativo. Il "messaggio" può essere lungo fino a 80 caratteri.

Se vengono battuti meno dati di quanti richiesti il calcolatore risponde con "??" sino a che non siano forniti tutti i dati voluti dal programma. Se vengono battuti dati in eccedenza essi saranno ignorati e apparirà il messaggio $EXTRA IGNORED.

Esempio:

Istruzione	Risposta utente	Risultato
10 INPUT A,B,C$	?123,456,NOW	A=123, B=456, C$="NOW"
10 INPUT A,B,C$	?123 ??456 ??NOW	A=123 B=456 C$="NOW"
10 INPUT A,B,C$	?NOW ?REDO FROM START ?123 ?456 ?789	 A=123 B=456 C="789"
10 INPUT "A= ";A	A= ?123	A=123

Notate che siete obbligati a battere dati numerici in corrispondenza alle variabili numeriche, ma potete dare invece numeri e lettere in corrispondenza alle stringhe.

Attenzione: Se battete RETURN prima di battere alcun dato, il programma si interrompe e il calcolatore passa in modo immediato. Per proseguire l'esecuzione battete CONT dopo il messaggio READY.

INPUT

L'istruzione INPUT # Esterna (INPUT #) permette l'ingresso di uno o più dati da una periferica esterna identificata da un numero di file logico.

Formato:

INPUT # If var [,var, ..., var]

I dati provenienti dal file logico If sono assegnati alle variabili "var". I dati devono corrispondere in numero e tipo alle variabili "var".

Se prima di aver ricevuto tutti i dati richiesti, si incontra la fine di un record si genera allora un errore di stato OUT OF DATA, ma il programma continua la sua esecuzione.

INPUT # e INPUT hanno le stesse caratteristiche salvo che la prima riceve i dati da un file logico. Anche INPUT # non visualizza messaggi di errore, ma modifica il valore della parola di stato. La parola di stato può poi essere interrogata dal programma stesso.

Le stringhe di dati in ingresso non possono essere più lunghe di 80 caratteri (79 più uno per il ritorno del carrello) perchè il buffer ha una capacità massima di 80 caratteri. Le virgole e i segni di ritorno del carrello sono trattati come separatori e non sono passati al programma come dati.

INPUT # può essere usata solo in modo differito.

Esempio:

`1000 INPUT#10,A`	Richiede in ingresso un valore numerico dal file numero 10; il valore viene associato alla variabile A
`946 INPUT#12,A$`	Richiede in ingresso una stringa dal file numero 12; la stringa viene associata alla variabile A$
`900 INPUT#5,B,C$`	Richiede in ingresso due valori dal file numero 5. Il primo valore deve essere numerico e viene associato alla variabile B, il secondo deve essere una stringa e viene associato alla variabile C$

LET =

L'istruzione di assegnazione LET =, o più semplicemente =, permette di attribuire un valore ad una variabile.

Formato:

{ (spazio) / LET } var = dato

Il valore di "dato", che può essere anche una espressione viene assegnato alla variabile "var".

La parola LET è facoltativa e spesso viene omessa.

Esempio:

```
10 A=2
450 C$="◥"

300 M(1,3)=SGN(X)
310 XX$(I,J,K,L)="STRINGALONG"
```

LIST

L'istruzione LIST visualizza il programma presente in memoria.

Formato:

LIST { (spazio) / linea / $linea_1$ — $linea_2$ / — linea / linea — }

È possibile visualizzare solo una parte di un programma indicando i numeri di linea estremi.

Esempio:

`LIST`	Lista l'intero programma
`LIST 50`	Lista la sola linea 50
`LIST 60-100`	Lista dalla linea 60 alla 100 incluse
`LIST -140`	Lista tutto il programma sino alla linea 100
`LIST 20000-`	Lista dalla linea 20000 sino alla fine del programma

In alcuni casi le linee di un programma sono visualizzate in forma più intelleggibile:

1. La forma abbreviata di "?" viene riportata per esteso con PRINT. Per esempio:

```
?A          diviene          PRINT A
```

2. Gli spazi davanti ai numeri di linea sono eliminati. Per esempio:

```
 50 A=1                      diviene        50 A=1
100 A=A+1                    diviene        100 A=A+1
```

3. Uno spazio viene sempre inserito dopo il numero di linea. Per esempio:

```
55A=B-2                      diviene        55 A=B-2
```

4. Le linee sono visualizzate a iniziare dalla colonna 2 anzichè dalla 1.

LIST è sempre usata in modo immediato. Se viene usata in modo differito essa lista il programma e poi pone il calcolatore in modo immediato. Se tentate di continuare l'esecuzione del programma con CONT otterrete la ripetizione di LIST indefinitivamente.

Stampa del listato di un programma

Per avere la stampa di un listato, anzichè la sua visualizzazione, è necessario aprire un file logico su una stampante prima di dare il comando LIST. In modo immediato potete dare questa sequenza di istruzioni:

```
OPEN 4,4      Apre la stampante specificando il file logico 4
CMD 4         Commuta l'uscita sulla stampante
LIST          Stampa il listato del programma
PRINT#4       Riporta l'uscita verso il display
CLOSE 4
```

LOAD

L'istruzione LOAD carica un programma da una periferica esterna nella memoria centrale. (Vedere anche DLOAD).

Formato per le unità a cassette:

```
LOAD ["nomefile"] [,dev]
```

Viene caricato il programma denominato "nomefile" dalla cassetta dev. Se il parametro dev non viene specificato viene imposta la cassetta principale numero 1. Se non viene specificato il nome del programma, viene allora caricato il primo programma incontrato sulla cassetta.

Per l'uso delle cassette vedere il capitolo 2.

Esempio:

`LOAD`	Carica in memoria il primo programma che incontra sulla cassetta 1. Se la cassetta si trova a metà di un programma, viene letto il programma successivo
`LOAD "",2`	Carica in memoria il programma successivo che incontra sulla cassetta 2
`LOAD "EGOR"`	Ricerca e carica il programma EGOR dalla cassetta 1
`N$="WHEE!LS"` `LOAD N$`	Ricerca e carica il programma WHEE!LS dalla cassetta 1
`LOAD "X"`	Ricerca e carica il programma X dalla cassetta 1

Formato per le unità a disco:

LOAD "dr: nomefile", dev

Viene caricato il programma "nomefile" dal dischetto nel drive dr. dev è il numero della periferica che per le unità a dischetto è sempre 8. Se dev manca viene imposto il valore 1 che corrisponde invece alle unità a cassette.

Se ponete al posto di "nomefile" un asterisco "*" verrà caricato il primo programma incontrato.

Per l'uso dei dischetti vi rimandiamo al capitolo 2.

Esempio:

`LOAD"0:*",8`	Carica il primo programma trovato sul dischetto nel drive 0
`LOAD"0:FIREBALL",8`	Ricerca e carica il programma FIREBALL dal dischetto nel drive 0
`T$="0:METEOR"` `LOAD T$,8`	Ricerca e carica il programma METEOR dal dischetto nel drive 0

Quando LOAD è eseguita in modo immediato, il calcolatore CBM esegue anche un comando CLR prima di caricare il programma. Una volta che un programma è in memoria può essere listato, modificato o eseguito.

L'istruzione LOAD può essere usata anche in modo differito per ottenere il concatenamento di più programmi. Quando LOAD viene eseguita ferma l'esecuzione del programma corrente e fa caricare un altro programma. In questo caso però non viene eseguita la CLR per cui il vecchio programma passa tutte le sue variabili a quello nuovo.

Quando LOAD è eseguita in modo differito, per caricare un programma da un dischetto, non vengono visualizzati i messaggi di caricamento. Se fa riferimento invece ad una unità a cassette, i messaggi di caricamento sono soppressi se il tasto PLAY del registratore è abbassato. Se invece il tasto PLAY è alzato viene dato il messaggio PRESS PLAY ON TAPE # 1 per poter eseguire la lettura della cassetta.

Impiego di LOAD per visualizzare la directory di un dischetto

In BASIC 4.0 si usa l'istruzione DIRECTORY per visualizzare la directory di un dischetto. Nelle versioni precedenti di BASIC è necessario invece caricare e listare il programma che ha nome "$0" (per un dischetto nel drive 0) e "$1" (per un dischetto nel drive 1).

Esempio:

```
LOAD "$0",8
SEARCHING FOR $0
LOADING
READY
LIST
```

NEW

L'istruzione NEW elimina il programma corrente dalla memoria.

Formato:

```
NEW
```

Quando l'istruzione NEW viene eseguita tutte le variabili sono inizializzate a zero o al valore nullo mentre la zona di memoria riservata alle variabili con indici, viene resa libera. Il programma presente in memoria non viene fisicamente cancellato, ma i suoi puntatori sono inizializzati; questo ai fini della sua esecuzione ha lo stesso effetto della cancelllazione. L'istruzione NEW viene automaticamente eseguita quando si effettua una operazione LOAD in modo immediato.

Prima di caricare un nuovo programma dalla tastiera dovete dare in modo immediato un comando NEW, altrimenti il nuovo e il vecchio programma si sovrappongono.

Esempio:

```
NEW
```

NEW è sempre eseguita in modo immediato; se la ponete in un programma non otterrete altro che la distruzione dello stesso programma.

ON ... GOSUB

L'istruzione ON ... GOSUB permette il salto condizionato ad una particolare subroutine, scelta tra più subroutine, in dipendenza del valore assunto da una variabile.

Formato:

ON byte GOSUB $linea_1$ [,$linea_2$, ..., $linea_n$]

ON ... GOSUB ha lo stesso formato di ON ... GOTO a cui vi rimandiamo per le

regole di salto. "byte" viene calcolato e riportato ad un valore intero.

Se byte = 1 viene richiamata la subroutine in $linea_1$, se byte = 2 viene richiamata la subroutine in $linea_2$ e così via. Il ritorno dalla subroutine avviene sempre all'istruzione successiva alla ON ... GOSUB.

Questa istruzione viene normalmente eseguita in modo differito. Può essere eseguita in modo immediato purchè esistano in quel momento in memoria le varie subroutine.

Esempio:

```
10 ON A GOSUB 100,200,300
```

ON ... GOTO

L'istruzione ON ... GOTO permette un salto condizionato in dipendenza del valore assunto da una variabile.

Formato:

ON byte GOTO $linea_1$ [,$linea_2$, ..., $linea_n$]

"byte" è calcolato e riportato ad un valore intero.

Se byte = 1 il programma salta alla $linea_1$, se byte = 2 salta alla $linea_2$ e così via. Se byte = 0, oppure se il numero di linea richiamato manca, non avviene il salto e il programma continua alla prossima istruzione. Ogni volta che il salto non può avvenire il programma continua alla prossima istruzione che può essere sulla stessa linea di ON ... GOTO, ma separata dai due punti, oppure sulla linea successiva.

Se byte ha un valore non zero, ma fuori dei limiti consentiti il programma si interrompe e appare un messaggio di errore. Si possono indicare tanti numeri di linea sino a riempire una riga di 80 caratteri.

ON ... GOTO si usa normalmente in modo differito. Si può usare anche in modo immediato purchè esistano le corrispondenti linee di programma in memoria.

Esempio:

```
40 A=B<10
50 ON A+2 GOTO 100,200
```

Salta all'istruzione 100 se A è vero (−1) o all'istruzione 200 se A è falso (0)

```
50 X=X+1
60 ON X GOTO 500,600,700
```

Salta all'istruzione 500 se X = 1; all'istruzione 600 se X = 2; all'istruzione 700 se X = 3. Prosegue alla linea successiva se X > 3

OPEN

L'istruzione OPEN apre un file logico e prepara la corrispondente periferica per le operazioni di ingresso o uscita. (Vedere anche DOPEN).

Formato per le unità a cassette:

OPEN lf [,dev] [,sa] [,"nomefile"]

Il file denominato "nomefile" presente sull'unità a cassette "dev" viene aperto per una operazione specificata dall'indirizzo secondario "sa". Le operazioni con il file avvengono tramite il file logico "lf".

Se manca "nomefile" viene selezionato il primo file incontrato sulla cassetta. Se manca "dev" viene imposto il valore 1. Se manca l'indirizzo secondario "sa" viene imposto il valore 0 che permette solo operazioni di lettura del file. Se sa = 1 il file è aperto in scrittura; se sa = 2 il file è aperto in scrittura, ma in coda ad esso sarà posto un EOT al momento della chiusura.

Esempio:

`OPEN 1`	Apre il file logico 1 sulla cassetta 1 (default) per operazioni di lettura (default) per il primo file incontrato (non essendovi alcuna specifica di nome)
`OPEN 1,1`	Come sopra
`OPEN 1,1,0`	Come sopra
`OPEN 1,1,0,"DAT"`	Come sopra, ma per il file DAT
`OPEN 3,1,2`	Apre il file logico 3 sulla cassetta 1 per operazioni di scrittura con un EOT (End Of Tape) alla fine del file. Il file non ha nome
`OPEN 3,1,2,"PENTAGRAM"`	Come sopra, ma il file ha nome PENTAGRAM

Formato per le unità a disco:

OPEN lf, dev, sa, "dr: nomefile, tipo [,accesso]"

Il file denominato "nomefile" presente sul dischetto nel drive "dr" viene aperto e posto in correlazione con il file logico "lf". "tipo" determina il tipo di file: SEQ per sequenziale, PRG per programma e USR per diretto. Se il file è di tipo sequenziale allora "accesso" deve indicare WRITE in caso di scrittura e READ in caso di lettura. Nel caso di programmi o file ad accesso diretto non si deve indicare l'"accesso".

Un file sequenziale già esistente può essere aperto in scrittura se "dr" viene preceduto dal segno @. Il contenuto di tale file viene completamente sostituito dai nuovi dati.

"dev"deve essere presente e indicare il numero della periferica; ricordiamo che deve essere 8 per le unità a disco. Se manca viene imposto il valore 1 che corrisponde invece alle unità a cassette.

I file di dati devono avere l'indirizzo secondario compreso tra 2 e 14. L'indirizzo 15 seleziona il canale di comando; gli indirizzi 0 e 1 sono riservati per i file di programmi (0 per caricare un programma e 1 per salvarlo).

Esempio:

`OPEN 1,8,2,"0:DAT,SEQ,READ"`	Apre il file logico 1 sul dischetto nel drive 0. Il file DAT verrà letto
`OPEN 5,8,3,"1:NEWFILE,SEQ,WRITE"`	Apre il file logico 5 sul dischetto nel drive 1. Il file NEWFILE verrà scritto
`OPEN 4,8,4,"@1:NEWFILE,SEQ,WRITE"`	Apre il file logico 4 sul dischetto nel drive 1. Il contenuto del file NEWFILE verrà riscritto

Per la gestione dei file vedere il capitolo 6.

POKE

L'istruzione POKE memorizza un byte in una posizione di memoria.

Formato:

```
POKE memoind, byte
```

Un valore intero compreso tra 0 e 255 viene caricato nella posizione di memoria "memoind".

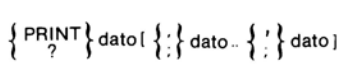
Esempio:

`10 POKE 1,A`	Forza il valore della variabile A nella posizione di memoria 1
`POKE 32768,ASC("A")-64`	Forza il valore 1 (ottenuto da ASC ("A") − 64) nella posizione di memoria 32768

PRINT

L'istruzione PRINT permette di visualizzare o stampare dati.

Formato:



Formato delle visualizzazioni

I numeri compresi tra 0.01 e 999999999. sono rappresentati nella forma standard. I numeri esterni a tale intervallo sono rappresentati invece nella forma esponenziale. Tutti i numeri sono preceduti dal segno e seguiti da un carattere di separazione.

Il segno (+) viene indicato come spazio vuoto. Le stringhe vengono visualizzate così come sono. Se più variabili sono presenti nella lista di una PRINT valgono le seguenti regole:

Prima variabile. La prima variabile viene visualizzata ad iniziare dalla posizione attuale del cursore. Le variabili successive sono visualizzate subito di seguito o spostate a seconda che siano precedute dalla virgola o dal punto e virgola.

Nuova linea. Se l'ultima variabile non è seguita ne da una virgola ne da un punto e virgola, allora dopo tale variabile il cursore va a capo.

Tabulazione. La presenza della virgola fa iniziare la variabile successiva nel punto di tabulazione fisso seguente (cioè alle colonne 1, 11, 21 e 31 per lo schermo a 40 colonne, oppure anche alle colonne 41, 51, 61 e 71 per quello a 80 colonne). Se una virgola precede la prima variabile, allora verrà subito eseguito un salto di tabulazione prima di visualizzare la prima variabile.

Visualizzazione continua. Il punto e virgola permette la visualizzazione continua dei dati. Le stringhe sono effettivamente poste in successione continua, mentre i numeri hanno sempre uno spazio separatore.

Esempio:

```
40 PRINT A
40 PRINT A,B,C
40 PRINT A;B;C
40 PRINT, A;B;C
40 PRINT "NUMBERS",A;B;C
40 PRINT "NUM";"BER";
41 PRINT "S",A;B;C
```

PRINT

L'istruzione PRINT # Esterna permette l'uscita di dati dal calcolatore verso una periferica esterna identificata da un numero di file logico (unità a cassette, a disco o stampanti).

Formato:

PRINT # lf, dato; c$; dato; c$; ... dato

I dati riportati nella lista della PRINT # sono scritti sulla periferica selezionata dal numero lf di file logico.

I caratteri separatori nell'istruzione PRINT # devono rispettare regole molto precise di cui diamo, qui di seguito, un breve sommario. (Per una descrizione dettagliata vedere il capitolo 6).

PRINT # verso un file su cassette

Ogni stringa o dato numerico scritto su cassetta deve essere seguito da un carattere di ritorno del carrello. Questo carattere di ritorno del carrello è posto automaticamente se la variabile nella lista della PRINT # è unica. Se nella lista vi sono più variabili è necessario porre un carattere c$ che forzi un ritorno del carrello. Potete usare per esempio CHR$(13) oppure una stringa che in precedenza sia stata posta eguale a CHR$(13).

PRINT # verso un file su dischetto

Le regole più sopra riportate per i file su cassetta valgono anche per i file su dischetto salvo una unica eccezione: le variabili di stringa possono essere separate da virgole (CHR$(44)). Le virgole, come già il ritorno del carrello, devono essere inserite mediante una stringa c$. Le stringhe scritte con la virgola, come elemento separatore, devono poi essere lette con una unica istruzione INPUT #. Infatti la INPUT # legge un testo tra un carattere di ritorno del carrello e il successivo.

PRINT # verso una stampante

Quando l'istruzione PRINT # si riferisce ad una stampante la stringa di separazione c$ deve essere CHR$(29). Non devono essere posti, inoltre, segni di interpunzione tra c$ e i dati come indicato nel formato riportato più sopra.

Attenzione: la forma? # non può sostituire PRINT #.

In BASIC < 3.0 l'istruzione PRINT # termina ogni linea con il ritorno del carrello. In BASIC 4.0 questo si verifica solo per i numeri di file superiori a 127. Alcune stampanti, non della Commodore, richiedono sempre un carattere di ritorno del carrello al termine di una riga. Se avete una di queste stampanti e lavorate in BASIC 4.0, vi basterà scegliere numeri di file superiori a 127 oppure porre esplicitamente il carattere di ritorno del carrello.

Esempio:

`100 PRINT#1,A`	Fa uscire la variabile numerica A e un carattere di ritorno del carrello sul file logico 1
`200 PRINT#4,A$`	Fa uscire la stringa A$ e un carattere di ritorno del carrello sul file logico 4
`300 PRINT#10,B%;",";C$`	Fa uscire la variabile numerica B%, una virgola, la stringa C$ e un carattere di ritorno del carrello sul file logico 10
`10 OPEN 1,1,2`	Apre il file logico 1 sulla cassetta 1 per operazioni di scrittura
`20 PRINT#1,"HI"`	Fa uscire la stringa "HI" sul file logico 1

L'istruzione PRINT # permette di eseguire alcune importanti operazioni sui dischi che qui di seguito riassumiamo. In BASIC 4.0 esistono per queste operazioni delle istruzioni separate.

I file su disco devono essere chiusi prima di poter eseguire queste operazioni.

COPY

Con PRINT # potete copiare e/o unire più file. (Vedere COPY e CONCAT in BASIC 4.0).

Formato:

PRINT # lf, "C [OPY] d: filedest = s: filesorgente [,s: filesorgente ...]"

Sino a quattro file sorgente possono essere copiati e concatenati in un file destinazione. I file sorgente rimangono inalterati. "s" è il drive del dischetto sorgente. "d" il drive del dischetto destinazione. Se i file sorgente sono più di uno, essi saranno copiati nell'ordine con cui sono riferiti nella PRINT #.

Esempio:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`PRINT#1,"C1:FILE1=C0:FILE0"`	Copia il file FILE 0 nel drive 0 del disco, con il nuovo nome FILE1, nel drive 1
`PRINT#1,"C0:NEWFIL=C1:FILEA,C0:FILEB"`	Un nuovo file con nome NEWFILE nel drive 0, viene creato concatenando il file FILEB nel drive 0 alla file del file FILEA nel drive 1

DUPLICATE

Con PRINT # potete duplicare un dischetto. (Vedere BACKUP in BASIC 4.0).

Formato:

PRINT # lf, "D [UPLICATE] d = s"

Il dischetto nel drive "d" diviene la copia del dischetto nel drive "s". Sia il nome che il numero di identificazione, come anche tutti i file, sono ricopiati esattamente.

Prima di duplicare un dischetto è consigliabile coprire la sua tacca di protezione da scrittura. Così se per errore invertite i due dischetti, oppure la s con la d, otterrete al massimo una segnalazione di errore e non distruggerete il dischetto che volevate copiare.

Esempio:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`.`	
`PRINT#1,"D0=1"`	Il dischetto nel drive 1 viene duplicato; il duplicato è generato nel drive 0
`.`	
`PRINT#1,"DUPLICATE0=1"`	Come sopra

INITIALIZE

Usate PRINT # per inizializzare un dischetto. Questa operazione non è necessaria se lavorate con il DOS versione 2.0 o superiori.

Formato:

```
PRINT # lf, "I [NITIALIZE] [dr]
```

Il dischetto nel drive dr viene inizializzato. Se dr manca vengono inizializzati i dischetti in ambedue i drive.

Le versioni di DOS precedenti alla 2.0 richiedono che i dischetti siano sempre inizializzati prima del loro normale uso. È importante ricordare che il BASIC 3.0, e le versioni precedenti, lavorano con questi DOS.

Il DOS 2.0 e le versioni seguenti inizializzano invece automaticamente i dischetti quando vengono inseriti nei drive. Il BASIC 4.0 deve lavorare con il DOS versione 2.0 o seguenti.

Il processo di preparazione di un dischetto effettua anche la sua inizializzazione.

Esempio:

```
OPEN 1,8,15                 Apre il canale di comando del dischetto
.
PRINT#1,"I"                 Inizializza i dischetti nei drive 0 e 1
.
PRINT#1,"INITIALIZE1"       Inizializza il dischetto nel drive 1
```

NEW

Con questa istruzione potete preparare e dare formato ad un dischetto nuovo oppure ad uno vecchio che intendete riusare. (Vedere anche HEADER in BASIC 4.0).

Formato:

```
PRINT # lf, "N [EW] dr: nomedisco, vv"
```

Viene preparato il dischetto "nomedisco" con il numero d'identificazione "vv" nel drive dr. In questa fase vengono segnati a software i settori e vengono inizializzati la directory e la mappa di disponibilità del blocchi BAM.

Il nome del disco e il numero di identificazione appariranno sempre in testa alla directory.

Esempio:

```
OPEN 1,8,15                 Apre il canale di comando del dischetto
.
PRINT#1,"NO:NEWDATA,02"     Viene preparato un dischetto nel drive 0. Il dischetto prende il
                            nome NEWDATA e il numero 02
```

Quando preparate un vecchio dischetto potete mantenere il vecchio numero d'identificazione, ma dargli un nuovo nome; oppure potete mantenere sia il vecchio nome che il numero d'identificazione. Supponete per esempio di avere un dischetto

con il nome NEWDATA e il numero 02. Le seguenti istruzioni di preparazione sono valide:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`PRINT#1,"NEW0"`	Prepara un vecchio dischetto nel drive 0 mantenendo il suo nome e numero
`PRINT#1,"N1:NEWDISK"`	Prepara un vecchio dischetto nel drive 1. Cambia in NEWDISK il nome, ma conserva il vecchio numero
`PRINT#1,"N1:NEWDATA,01"`	Come sopra, ma con il nuovo nome NEWDATA e il nuovo numero 01

La seguente istruzione è invece illegale:

```
PRINT#1,"N0:02"
```

essa infatti tenta di dare un nuovo numero pur mantenendo il vecchio nome.

RENAME

Con questa istruzione potete cambiare il nome di un file (Vedere anche RENAME in BASIC 4.0).

Formato:

PRINT # lf, "R [ENAME] dr: nomenuovo = nomevecchio"

Il file che portava il nome "nomevecchio" viene denominato con il nome "nomenuovo".

Esempio:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`PRINT#1,"R1:BACKUP=CURRENT"`	Cambia il nome del file CURRENT, nel drive 1, nel nuovo nome BACKUP

SCRATCH

Con l'istruzione SCRATCH potete cancellare uno o più file da un dischetto. (Vedere anche SCRATCH in BASIC 4.0).

Formato:

PRINT # lf, "Sdr: nomefile [,dr: nomefile]"

Con una istruzione PRINT # potete cancellare da uno a tutti i file di un dischetto oppure anche di due dischetti.

Per ogni file da cancellare indicate il nome e il numero del drive in cui si trova il dischetto.

Se dovete cancellare più file che inizino con le stesse lettere potete usare l'asterisco (*). Così per esempio se volete cancellare tutti i file che inizino con le stesse lettere FILE è sufficiente che poniate nella stringa del nome "FILE*". Se ponete

invece "F*" verranno cancellati tutti i file che iniziano per F. Con "*" cancellerete indistintamente tutti i file di un dischetto.

L'uso dell'asterisco è lo stesso che potete fare con OPEN, DOPEN e DLOAD.

Accanto all'asterisco potete usare anche il punto interrogativo (?). Per esempio se nella stringa del nome scrivere "FILE?DATI" verranno cancellati tutti i file che abbiano un qualunque carattere nella stessa posizione del punto interrogativo e le altre lettere eguali. Con "F???K" vengono cancellati tutti i file con i nomi di cinque caratteri che iniziano con F e terminano con K.

Esempio:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`PRINT#1,"S0:FILENAME"`	Cancella il file FILENAME nel drive 0
`PRINT#1,"S0:FILENAME,1:NEWFILE"`	Come sopra, ma cancella anche il file NEWFILE nel drive 1
`PRINT#1,"S0:FILENAME,0:NEW*"`	Come sopra, ma cancella anche tutti i file i cui nomi iniziano con le lettere NEW nel drive 0
`PRINT#1,"S1:A???"`	Cancella tutti i file i cui nomi iniziano con la lettera A seguita da tre caratteri qualunque nel drive 1
`PRINT#1,"S0:*"`	Cancella tutti i file nel drive 0

VALIDATE

Con VALIDATE potete convalidare (o riordinare) un dischetto. (Vedere anche COLLECT in BASIC 4.0).

Formato:

PRINT # lf, "V [ALIDATE] [dr]"

Viene convalidato il dischetto nel drive dr. Se dr manca viene convalidato il dischetto posto nel drive selezionato per ultimo.

Con questa operazione viene riordinata la mappa di disponibilità dei blocchi BAM. Tutti i file ancora aperti o non correttamente chiusi vengono cancellati e i loro settori (o blocchi) diventano liberi.

Non convalidate un dischetto che contenga file ad accesso diretto: tali file verrebbero cancellati! (In realtà i blocchi di questi file verrebbero resi subito disponibili per la scrittura di un qualsiasi altro file).

Se durante l'operazione di convalida avviene un errore di lettura, l'operazione stessa è interrotta e il dischetto rimane nel suo stato iniziale.

Un dischetto deve essere inizializzato dopo la convalida.

Esempio:

`OPEN 1,8,15`	Apre il canale di comando del dischetto
`PRINT#1,"I0"`	Inizializza il dischetto nel drive 0
`PRINT#1,"V0"`	Convalida il dischetto nel drive 0

READ

L'istruzione READ assegna i valori, contenuti nell'istruzione DATA, alle variabili contenute nella sua lista.

Formato:

```
READ var [,var, ..., var]
```

L'istruzione READ equivale a più istruzioni di assegnazione LET.

Una o più istruzioni READ devono avere in corrispondenza una o più istruzioni DATA. Le variabili nelle liste delle READ devono corrispondere in numero e tipo alle costanti nelle liste delle DATA. Le variabili di stringa possono accettare qualunque tipo di costante, mentre le variabili numeriche possono accettare solo costanti numeriche.

Le istruzioni READ e DATA possono differire in numero, ma deve essere disponibile una costante per ogni variabile riportata in una READ.

Vi possono essere più costanti nelle DATA che variabili nelle READ, ma non viceversa. In quest'ultimo caso si avrebbe la segnalazione di errore? OUT OF DATA.

READ è generalmente eseguita in modo differito. Può essere eseguita in modo immediato purchè esista una corrispondente DATA in un programma memorizzato da cui prelevare le costanti.

Esempio:

```
10 DATA 1,2,3
20 READ A,B,C
```

Si ottiene dopo l'esecuzione: A = 1, B = 2, C = 3

```
150 READ C$,D,F$
160 DATA STR
170 DATA 14.5,"TM"
```

Si ottiene dopo l'esecuzione: C$ = "STR", D = 14.5, F$ = "TM"

RECORD (BASIC 4.0)

L'istruzione RECORD permette di portare un puntatore su ogni byte (carattere) di ogni record di un file relativo. L'istruzione RECORD viene usata prima di GET#, INPUT# o PRINT#.

Formato:

```
RECORD # lf, rno [,bno]
```

L'istruzione RECORD permette di selezionare il byte numero bno del record rno del file lf.

Se l'istruzione RECORD pone il puntatore oltra la fine del file e una istruzione PRINT # tenta di scrivere un nuovo record, il file viene automaticamente esteso per comprendere questo nuovo record. Diversamente se una RECORD ha puntato oltre la fine del file e una istruzione INPUT # tenta di richiamare dei dati, si avranno in ingresso dei valori nulli e nella parola di stato ST comparirà un avviso di fine del file.

Esempio:

```
10 DOPEN#1,"DATAFILE",L20,6: REM RELATIVE FILE DATAFILE HAS 20 BYTES PER RECORD
20 RECORD#1,20,6: REM SELECT THE 6TH BYTE RECORD N0. 20
30 GET#1,A$:IF A$= THEN 30: REM LOAD THIS BYTE INTO A$
40 STOP
```

REM

L'istruzione REM permette di inserire ovunque in un programma dei commenti per motivi esplicativi e di documentazione.

Formato:

REM commento

dove il commento è una qualunque sequenza di caratteri che rientri in una riga di 80 colonne.

Le istruzioni REM sono riprodotte nei listati, ma ovviamente non vengono eseguite. Tali istruzioni possono essere poste ognuna su una linea di programma oppure in coda su una linea con più istruzioni.

Le REM non possono mai precedere altre istruzioni su una stessa linea, in quanto il calcolatore considera un commento tutto ciò che segue REM.

Le REM possono essere poste lungo il cammino logico di esecuzione di un programma e ad esse può anche essere indirizzato un salto.

Esempio:

```
10 REM *** * * * * * * * * * * * * * * * ****
20 REM ***PROGRAM EXCALIBUR***
30 GOTO 55:REM BRANCH IF OUT OF DATA
```

RENAME (BASIC 4.0)

L'istruzione RENAME permette di cambiare il nome di un file su un dischetto senza cambiarne il contenuto. (Vedere anche PRINT # RENAME).

Formato:

RENAME [dr] "nomevecchio" TO "nomenuovo" [ON Uz]

Il file "nomevecchio", presente sul dischetto nel drive dr, cambia il suo nome in "nomenuovo". Se il parametro dr manca viene imposto il valore 0.

Se avete delle difficoltà a cambiare il nome ad un file, provate allora ad eseguire prima l'operazione COLLECT e poi RENAME.

Attenzione: Prima di eseguire RENAME dovete aver chiuso il file.

Esempio:

```
RENAME "PET" TO "CBM"        Il nome del file PET nel drive 0 viene cambiato in CBM
RENAME D1, "ONE" TO "TWO"    Il nome del file ONE nel drive 1 viene cambiato in TWO
```

RESTORE

L'istruzione RESTORE riporta al valore iniziale il puntatore dell'istruzione DATA.

Formato:

```
RESTORE
```

L'istruzione RESTORE può essere data sia in modo differito che immediato.

Esempio:

```
10 DATA 1,2,N44
20 READ A,B,B$        A = 1, B = 2, B$ = "N44"
30 RESTORE
40 READ X,Y,Z$        X = 1, Y = 2, Z$ = "N44"
```

RETURN

L'istruzione RETURN permette il ritorno da una subroutine all'istruzione subito seguente a GOSUB.

Ogni subroutine deve terminare con una istruzione RETURN.

Formato:

```
RETURN
```

Esempio:

```
100 RETURN
```

Notate che l'istruzione RETURN permette il ritorno del controllo di un programma da una subroutine, mentre il tasto RETURN muove invece il cursore a capo all'inizio di una nuova linea. **I due comandi non hanno assolutamente niente in comune!**

RUN

RUN pone in esecuzione il programma in quel momento presente nella memoria centrale. RUN chiude ogni file eventualmente aperto e riporta tutte le variabili al valore zero o nullo.

Formato:

```
RUN linea
```

Quando RUN viene eseguita in modo immediato, il calcolatore effettua anche una CLR di tutte le variabili del programma e inizializza i puntatori dei dati in memoria (vedere anche RESTORE).

Se RUN specifica un numero di linea, il calcolatore esegue ugualmente CLR e

RESTORE, ma inizia l'esecuzione del programma dalla linea indicata.

L'istruzione RUN con numero di linea non può essere usata, dopo una interruzione del programma (break), per continuare l'elaborazione. In tal caso dovete usare CONT o GOTO.

RUN può essere usata in modo differito. In questo caso il programma si ripone in esecuzione con tutte le variabili nuovamente inizializzate.

Esempio:

`RUN`	Pone in esecuzione il programma presente in memoria
`RUN 1000`	Pone in esecuzione il programma presente in memoria a partire dalla linea 1000

SAVE

L'istruzione SAVE trascrive una copia del programma attualmente in memoria su una periferica esterna. (Vedere anche DSAVE).

Formato per le unità a cassette:

SAVE ["nomefile"] [,dev] [,sa]

L'istruzione SAVE scrive il programma attualmente in memoria, denominandolo "nomefile", sulla cassetta nel drive dev. Se dev manca viene imposto il valore 1, cioè viene scelta la cassetta principale. Se "nomefile" è presente viene scritto in testa al programma. Se l'indirizzo secondario sa non è zero, allora in coda al programma salvato viene posto un segno di fine file EOF.

Sebbene non sia strettamente necessario indicare tutti i parametri dell'istruzione SAVE, per salvare un programma su cassetta, è preferibile indicare sempre il nome per rendere poi più facile la ricerca del programma. Un programma registrato su cassetta può essere riletto tramite il suo nome o ricercando la sua posizione fisica sul nastro.

L'istruzione SAVE è comunemente usata in modo immediato sebbene possa anche essere usata in modo differito.

Per l'uso dettagliato delle unità a cassette vi rimandiamo al capitolo 2.

Esempio:

`SAVE`	Trascrive il programma presente in memoria sulla cassetta 1
`SAVE "RED"`	Come sopra, ma gli pone il nuovo nome "RED"
`A$="RED"` `SAVE A$`	Come sopra
`SAVE "BLACKJACK",2,1`	Trascrive il programma presente in memoria sulla cassetta 2 e gli pone il nome BLACKJACK. In coda al programma registra il segno di fine del file EOF.

Formato per le unità a disco:

```
SAVE "dr: nomefile", dev
```

L'istruzione SAVE scrive il programma attualmente in memoria, sul dischetto posto nel drive dr. Il programma prende il nome "nomefile". Il parametro dev deve essere presente e avere il valore 8. Se dev manca viene imposto il valore 1 che si riferisce invece alla unità principale a cassette.

Il nome assegnato "nomefile" non deve già esistere sul dischetto, altrimenti viene segnalato un errore di sintassi. Tuttavia se usate versioni di DOS 2.0 o superiori, potete sostituire un file di programma inserendo il carattere @ davanti a dr nell'istruzione SAVE. In tal caso viene cambiato il contenuto di un programma, ma non il suo nome.

SAVE è normalmente usata in modo immediato anche se può essere inserita in un programma.

Per l'uso dettagliato dei dischetti vedere il capitolo 2.

Esempio:

`SAVE "0:BLACKJACK",8`	Trascrive il programma presente in memoria sul dischetto nel drive 0 e gli pone il nuovo nome BLACKJACK
`SAVE "@0:BLACKJACK",8`	Trascrive il programma in memoria sul dischetto nel drive 0 sostituendo il contenuto di BLACKJACK

SCRATCH (BASIC 4.0)

L'istruzione SCRATCH cancella un singolo file da un dischetto. (Vedere anche PRINT # SCRATCH).

Formato:

```
SCRATCH [Dd,] "nomefile" [ON UZ]
```

Viene cancellato il file "nomefile" sul dischetto nel drive d. Se il parametro Dd manca viene imposto il valore 0. (Anche in questo caso è possibile usare l'asterisco e il punto interrogativo per denominare i file).

L'istruzione SCRATCH può essere eseguita sia in modo immediato che differito. In modo immediato è impiegata quando si debbano fare delle operazioni di gestione del dischetto ("housekeeping"). Data l'importanza di questa operazione il calcolatore chiederà la conferma ARE YOU SURE? a cui dovete rispondere YES <CR> oppure Y <CR>. Diversamente il file non sarà cancellato.

Se questa istruzione viene eseguita in modo differito non appaiono messaggi di avviso o di conferma. Spesso in un programma è necessario usare dei file di dati temporanei che devono poi essere cancellati per poter rieseguire lo stesso programma. Se non venissero cancellati, durante una successiva esecuzione si avrebbe l'errore FILE EXISTS.

I file devono essere chiusi prima di poter essere cancellati. Se per errore tentate di cancellare un file ancora aperto, il calcolatore può effettuare erroneamente opera-

zioni irregolari sul dischetto.

Se lavorate in DOS 2.0 è consigliabile eseguire una istruzione COLLECT prima di cancellare qualunque file. (Vedere COLLECT).

Esempio:

`SCRATCH D0,"DUMMY1"`	Cancella il file DUMMY1 sul dischetto nel drive 0
`SCRATCH "DUMMY1"`	Come sopra
`SCRATCH D1,"FILE1"`	Cancella il file FILE1 sul dischetto nel drive 1

STOP

L'istruzione STOP fa sospendere l'esecuzione di un programma; il calcolatore ritorna in modo immediato e un messaggio di interruzione viene dato sullo schermo.

Formato:

```
STOP
```

Esempio:

`655 STOP`	Ferma l'esecuzione e fa apparire il messaggio BREAK IN 655

VERIFY

L'istruzione VERIFY confronta il contenuto di un programma registrato con quello presente in memoria.

Formato per le unità a cassette:

```
VERIFY ["nomefile"] [,dev]
```

Il programma attualmente in memoria viene confrontato con il file di programma che ha nome "nomefile" presente sulla cassetta inserita nell'unità dev. Se dev manca viene imposto il valore 1 corrispondente alla cassetta principale. Se manca "nomefile" viene selezionato il primo programma incontrato sulla cassetta.

È importante che verifichiate sempre un programma dopo averlo salvato.

L'istruzione VERIFY viene quasi sempre eseguita in modo immediato. Per l'impiego delle unità a cassette vi rimandiamo al capitolo 2.

Esempio:

`VERIFY`	Verifica il programma successivo incontrato sul nastro
`VERIFY "CLIP"`	Verifica il programma CLIP sulla cassetta 1
`A$="CLIP"` `VERIFY A$`	Come sopra

Formato per unità a disco:

```
VERIFY "dr: nomefile", dev
```

Il programma attualmente in memoria viene confrontato con il file di programma che ha nome "nomefile" presente sul dischetto nel drive dr. Il parametro dr deve essere presente e avere il valore 8. Se dev manca viene imposto il valore 1 che corrisponde invece al registratore a cassette principale.

Se dovete verificare un programma che avete appena salvato, potete usare VERIFY con il seguente formato:

```
VERIFY "*",8
```

Vi consigliamo di verificare sempre un programma subito dopo averlo salvato.

L'istruzione VERIFY viene quasi sempre eseguita in modo immediato. Per l'impiego delle unità a dischetto vi rimandiamo al capitolo 2.

Esempio:

`VERIFY "*",8`	Verifica il programma appena salvato
`VERIFY"0:SHELL",8`	Ricerva e verifica il programma SHELL sul dischetto del drive 0
`C$="0:SHELL"` `VERIFY C$`	Come sopra

WAIT

L'istruzione WAIT interrompe l'esecuzione di un programma sino a che una certa posizione di memoria non assuma un determinato valore.

Formato:

```
WAIT memoind, maschera [,xor]
```

dove: maschera e xor sono due valori di un byte ciascuno.

L'istruzione WAIT funziona così:

1. Si individua il contenuto della posizione di memoria "memoind".
2. Fra tale valore e l'eventuale valore "xor" viene eseguita l'operazione logica XOR (OR Esclusivo). Se "xor" manca viene posto eguale a 0 così che l'operazione XOR lascia inalterato il valore contenuto in "memoind".
3. Il valore ottenuto al punto 2 viene unito (operazione AND) con il valore "maschera".
4. Se il risultato è 0, il ciclo ritorna al punto 1 così da effettuare l'attesa di WAIT.
5. Se il risultato è diverso da 0, il programma prosegue all'istruzione successiva a WAIT.

Il tasto STOP non può fermare l'esecuzione dell'istruzione WAIT.

FUNZIONI

Descriviamo qui di seguito in ordine alfabetico le funzioni del BASIC CBM.

Alcune funzioni, che sono disponibili solo con i calcolatori CBM 8000, saranno descritte nel paragrafo successivo.

ABS

ABS ritorna il valore assoluto di un numero, cioè senza il segno.

Formato:

ABS (daton)

Esempio:

```
A=ABS(10)                    Si ottiene A = 10
A=ABS(-10)                   Si ottiene A = 10
PRINT ABS(X),ABS(Y),ABS(Z)
```

ASC

ASC ritorna il numero in codice ASCII di uno specifico carattere.

Formato:

ASC (dato$)

Se la stringa dato$ è più lunga di un carattere, la funzione ASC ritorna il codice del primo carattere. Il valore ritornato è un numero che può essere usato in qualunque operazione aritmetica. Il codice ASCII è riportato nell'Appendice A.

Esempio:

```
?ASC("A")      Visualizza 65
N=ASC(B$)
X=ASC("S"),
?X             Visualizza 83 che è il valore ASCII di S
```

ATN

ATN ritorna l'arcotangente del suo argomento.

Formato:

ATN (daton)

ATN ritorna un valore espresso in radianti compresi nell'intervallo ± 17.

Esempio:

```
A=ATN(AG)
?180π*ATN(A)
```

CHR$

CHR$ ritorna la rappresentazione in stringa di un codice ASCII.

Formato:

CHR$ (byte)

CHR$ viene spesso usata per indicare caratteri che altrimenti non potrebbero essere inseriti in una stringa come per esempio il comando di ritorno del carrello e il simbolo di virgolette.

Esempio:

```
IF C$=CHR$(13) GOTO 10
```
Esegue il salto se C$ è il carattere di ritorno del carrello (CHR$(13))

```
?CHR$(34):"HOHOHO":CHR$(34)
```
Stampa gli otto caratteri "HOHOHO" (CHR$(34) rappresenta le virgolette)

COS

COS ritorna il valore coseno di un argomento.

Formato:

COS (daton)

Esempio:

```
AG=0.25
A=COS(AG)
```
Ad A viene assegnato il valore 0.968912422

DS (BASIC 4.0)

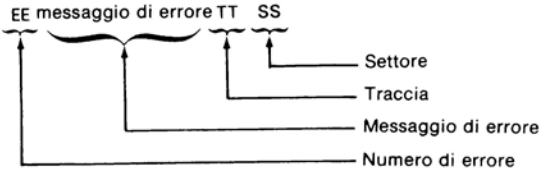
Ogniqualvolta la variabile DS viene richiamata da una istruzione BASIC, le viene assegnato automaticamente un valore intero che corrisponde alla operazione di accesso ad un disco più recente. Per l'interpretazione di questi valori interi vi rimandiamo alla Tabella 8-1.

Esempio:

```
20 IF DS<>0 THEN PRINT "ERROR":STOP
```

DS$ (BASIC 4.0)

Ogniqualvolta la stringa DS$ viene richiamata da una istruzione BASIC, viene ritornato lo stato del più recente accesso ad un disco con il seguente formato:



Nella Appendice B sono riportati i messaggi di errore per i dischetti.

Esempio:

```
20 IF DS>20 THEN PRINT DS$:STOP
```

Se DS ha il valore 1 significa che il file è stato cancellato; ogni altro valore inferiore a 20 non è un errore.

EXP

EXP ritorna il valore di e^{arg}. La base e è pari a 2.71828183.

Formato:

EXP(arg)

L'arg deve avere un valore compreso tra ±88.029691. Un numero maggiore darà luogo ad un superamento overflow con messaggio di errore, mentre un numero inferiore darà luogo ad un risultato nullo.

Esempio:

?EXP(0)	Visualizza 1
?EXP(1)	Visualizza 2.71828183
EV=EXP(2)	Pone EV = 7.3890561
EB=EXP(50.24)	Pone EV = 6.59105247 E+21
?EXP(88.0296919)	Il valore massimo consentito è 1.70141183 E+38
?EXP(-88.0296919)	Il valore minimo consentito è 5.877471 E-39
?EXP(88.029692)	Errore di superamento numerico (overflow)
?EXP(-88.029692)	Impone zero 0

FRE

FRE è una funzione di sistema, che colleziona tutti i byte di memoria non utilizzati in un unico blocco, chiamato "garbage collection", e ritorna il numero di byte liberi.

Formato:

FRE (arg)

arg è un argomento fittizio che può essere una stringa o un valore numerico. FRE viene normalmente usata nella lista di una PRINT in modo immediato.

Esempio:

```
?FRE(1)          Visualizza il numero di byte liberi
```

INT

INT ritorna la parte intera di un numero, arrotondandolo all'intero immediatamente inferiore.

Formato:

INT (org)

Per i numeri positivi INT equivale a togliere la parte frazionaria di un numero. Per quelli negativi equivale a togliere ancora la parte frazionaria, ma aggiungere anche —1. Notate che questa funzione non converte un numero da frazionario (5 byte) in intero (2 byte)!

Esempio:

```
A=INT(1.5)       Pone A = 1
A=INT(-1.5)      Pone A = -2
X=INT(-0.1)      Pone X = -1
```

Attenzione: I numeri con virgola nei calcolatori sono sempre una approssimazione di eventuali numeri reali, per cui alcune volte potrete ottenere un valore INT diverso da quello che vi aspettate. Per esempio se avete il numero 3.89999999 il suo valore INT è 3 e non 4 come qualcuno potrebbe erroneamente pensare.

```
?INT(3.89999999)
3
```

LEFT$

LEFT$ ritorna i caratteri più a sinistra della stringa.

Formato:

LEFT$ (arg$, byte)

byte indica quanti caratteri a sinistra devono essere estratti dalla stringa arg$.

Esempio:

```
?LEFT$("ARG",2)      Visualizza AR
A$=LEFT$(B$,10)      Visualizza i 10 caratteri più a sinistra della stringa B$
```

LEN

LEN ritorna il numero di caratteri della stringa arg$.

Formato:

LEN (arg$)

Esempio:

```
?LEN("ABCDEF")       Visualizza il numero 6
N=LEN(C$+D$)         Visualizza la lunghezza delle stringhe C$ e D$ concatenate
```

LOG

LOG ritorna il logaritmo naturale in base e dell'argomento.

Formato:

Se l'argomento è zero o negativo viene segnalato l'errore ILLEGAL QUANTITY ERROR.

Esempio:

```
?LOG(1)              Visualizza 0
A=LOG(10)            Pone A = 2.302585509
A=LOG(1E6)           Pone A = 13.8155106
A=LOG(X)/LOG(10)     Calcola il logaritmo in base 10
```

MID$

MID$ ritorna una parte di una stringa.

Formato:

MID$ (dato $, $byte_1$ [,$byte_2$])

Viene ritornata una parte centrale della stringa dato$. I due argomenti $byte_1$ e $byte_2$ determinano quale parte della stringa deve essere estratta. Tutti i caratteri

della stringa sono numerati da sinistra verso destra a iniziare con il numero 1 per il carattere più a sinistra. Il valore $byte_1$ determina quale è il primo carattere da estrarre. A iniziare da questo carattere $byte_2$ determina quanti caratteri verso destra sono da estrarre. Se $byte_2$ manca vengono estratti tutti i caratteri sino all'estremo destro.

Se un parametro esce dai limiti consentiti viene dato il messaggio ILLEGAL QUANTITY ERROR.

Esempio:

```
?MID$("ABCDE",2,1)      Visualizza B
?MID$("ABCDE",3,2)      Visualizza CD
?MID$("ABCDE",3)        Visualizza CDE
```

PEEK

PEEK ritorna il contenuto di una specifica posizione di memoria. PEEK è la funzione complementare di POKE.

Formato:

```
PEEK (memoind)
```

Tutte le posizioni di memoria possono essere lette mediante la funzione PEEK. (Talvolta le posizioni che contengono l'interprete BASIC sono protette contro la lettura per garantire la riservatezza del software di base del calcolatori CBM; in tal caso la funzione PEEK ritorna sempre il valore 0). Nel capitolo 7 sono indicate quelle zone della memoria che maggiormente vi interessa analizzare con la funzione PEEK.

Esempio:

```
?PEEK(1)                Visualizza il contenuto della posizione di memoria 1
A=PEEK(20000)
```

POS

POS ritorna la posizione attuale del cursore sullo schermo.

Formato:

```
POS (dato)
```

Il parametro "dato" è fittizio e può assumere qualunque valore. POS ritorna la posizione attuale del cursore. Se il cursore non è presente sullo schermo, viene egualmente ritornata la posizione dell'ultimo carattere visualizzato. La posizione dei caratteri inizia dall'estremo sinistro con il valore 0 e prosegue verso destra con valori crescenti. La funzione POS ritorna i valori tra 0 e 39, per lo schermo a 40 colonne, e tra 0 e 79 per lo schermo a 80 colonne.

Ricordatevi che la logica del calcolatore presume di lavorare sempre con linee di 80 caratteri anche se lo schermo ha 40 colonne. Se il calcolatore sta lavorando nella seconda metà di una linea quando riceve il comando POS, ritorna allora un valore tra 40 e 79 anche se lo schermo è a 40 colonne.

In alcuni casi la funzione POS può ritornare un valore compreso tra 0 e 255. Infatti se mediante concatenamento avete costruito una stringa lunga sino a 255 caratteri, e il calcolatore sta processando tale stringa, allora la funzione POS può ritornare un valore sino a 255 in corrispondenza del carattere della stringa che in quel momento è in elaborazione.

Esempio:

`?POS(1)`	Ritorna il valore 0 (inizio della linea)
`?"ABCABC";POS(1)`	Ritorna il valore 6 (se la stringa inizia al principio della linea)

RIGHT$

La funzione RIGHT$ ritorna i caratteri più a destra di una stringa.

Formato:

RIGHT$ (arg$, byte)

byte indica il numero di caratteri più a destra che vengono estratti dalla stringa arg$.

Esempio:

`RIGHT$(ARG,2)`	Visualizza RG
`MM$=RIGHT$(X$+"#",5)`	Pone in MM$ gli ultimi quattro caratteri di X$ e il segno #

RND

La funzione RND genera sequenze di numeri a caso compresi tra 0 e 1.

Formato:

RND (argn)	Ritorna un numero a caso
RND (—argn)	Memorizza un nuovo numero seme

Esempio:

`A=RND(-1)`	Pone un nuovo valore "seme" riferito a −1
`A=RND(1)`	Calcola un valore a caso successivo

L'argomento 0 viene considerato come un caso particolare in quanto non genera un numero a caso ne pone un nuovo seme. RND(0) usa il valore TI del tempo per introdurre un ulteriore elemento di aleatorietà nella funzione RND.

Un seme pseudo-random viene memorizzato mediante la funzione:

```
RND(-TI)          Pone un valore di "seme" pseudo-casuale
```

RND(0) può essere usata per porre un nuovo seme in modo aleatorio:

```
RND(-RND(0))      Pone un valore di "seme" casuale
```

Per una descrizione dettagliata della funzione RND vi rimandiamo al capitolo 5.

SGN

La funzione SGN determina se un numero è positivo, negativo o nullo.

Formato:

```
SGN (argn)
```

La funzione SGN ritorna +1 se il numero argn è positivo; ritorna 0 se argn è 0; ritorna −1 se argn è negativo.

Esempio:

```
?SGN(-6)          Visualizza -1
?SGN(0)           Visualizza 0
?SGN(44)          Visualizza 1
IF A<C THEN SA=SGN(X)
IF SGN(M)>= 0 THEN PRINT "POSITIVE NUMBER"
```

SIN

La funzione SIN ritorna il valore seno dell'argomento.

Formato:

```
SIN (argn)
```

Esempio:

```
A=SIN(AG)
?SIN(45*π/180)    Visualizza il seno di 45°
```

SPC

La funzione SPC muove il cursore a destra di un determinato numero di posizioni.

Formato:

```
SPC (byte)
```

La funzione SPC viene usata nella lista di una PRINT per muovere il cursore di

un certo numero di posizioni verso destra. Il testo su cui il cursore passa sopra non viene modificato.

La funzione SPC è analoga a quella TAB, ma con la differenza che SPC sposta di "byte" posizioni il cursore in senso relativo, mentre TAB porta il cursore in posizioni fisse, cioè in senso assoluto, come indicato dal suo argomento. (Vedere anche TAB).

SQR

La funzione SQR ritorna la radice quadrata di un numero positivo. Se l'argomento è negativo viene dato un messaggio di errore.

Formato:

```
SQR (argn)
```

Esempio:

```
A=SQR(4)          Pone A = 2
A=SQR(4.84)       Pone A = 2.2
?SQR(144E30)      Visualizza 1.2 E+16
```

ST

ST ritorna il valore attuale della parola di stato di I/O. Alla parola di stato viene assegnato un determinato valore in funzione dell'ultima operazione di ingresso/uscita.

Formato:

```
ST
```

I valori di stato sono riportati nella Tabella 8-3. Lo stato deve essere controllato dopo ogni operazione che acceda ad una periferica esterna. Per una descrizione dettagliata di questo argomento vi rimandiamo al capitolo 6.

Esempio:

```
10 IF ST <>0 GOTO 500              Visualizza 14.6
50 IF ST=4 THEN ?"SHORT BLOCK"
```

STR$

La funzione STR$ ritorna l'equivalente in stringa di un argomento numerico.

Formato:

```
STR$ (argn)
```

Questa funzione ritorna la stringa composta dallo stesso numero argn.

Esempio:

```
A$=STR$(14.6)        Visualizza 14.6
?A$
?STR$(1E2)           Visualizza 100
?STR$(1E10)          Visualizza 1 E+10
```

Tabella 8-3: Valori della parola di Stato

Posizione del bit in ST	Valore numerico di ST	Cassette in lettura	Verifica e LOAD da cassetta	Periferiche Collegate tramite IEEE-488
0	1			Superamento tempo in scrittura
1	2			Superamento tempo in lettura
2	4	Blocco corto	Blocco corto	
3	8	Blocco lungo	Blocco lungo	
4	16	Errore non recuperabile	Errore nell'operazione di Verify	
5	32	Errore di checksum	Errore di checksum	
6	64	Fine del file		Fine della identificazione
7	−128	Fine del nastro	Fine del nastro	Periferica non presente

SYS

SYS è una funzione di sistema che trasferisce il controllo del programma ad un sottosistema indipendente.

Formato:

```
SYS (memoind)
```

memoind è l'indirizzo di partenza del sottosistema.

Questo indirizzo memoind deve essere compreso tra 0 e 65535. Per una descrizione completa della funzione SYS vi rimandiamo al capitolo 7.

TAB

TAB porta il cursore nella posizione indicata dal suo argomento.

Formato:

TAB (argn)

TAB muove il cursore alla posizione argn + 1.

Esempio:

```
?"QUARK";SPC(10);"W"
QUARK          W

?"QUARK";TAB(10);"W"
QUARK      W
```

Questi due esempi mostrano la differenza tra SPC e TAB. SPC sposta il cursore di 10 posizioni dall'ultima, mentre TAB porta alla colonna 10 + 1

Come usare il tasto TAB

Molti calcolatori IBM hanno un tasto di tabulazione TAB. IL comando di questo tasto può essere inserito in una istruzione PRINT per impostare punti di tabulazione, per toglierli oppure per muovere il cursore a destra sino al più prossimo punto di tabulazione.

I punti di tabulazione sono posti, oppure rimossi, mediante il tasto TAB shiftato oppure con la funzione CHR$(9). Una tabulazione viene rimossa se il cursore si trova sopra una colonna in cui vi era già la tabulazione e viene ripetuto il comando di tabulazione. Altrimenti la tabulazione rimane: In altre parole se si tabula una posizione una volta sola essa rimane, ma se si ripete il comando essa viene cancellata.

La tabulazione può essere impostata oppure rimossa sia in modo immediato che differito. Per operare in modo immediato portate il cursore nella posizione desiderata e quindi premete il tasto TAB shiftato. Per operare in modo differito eseguite una PRINT, per muovere il cursore nella posizione desiderata, e quindi ponete il carattere CHR$(9).

Si possono porre sino a 80 punti di tabulazione, ma solo dopo RETURN diventano effettivi.

Il tasto TAB non shiftato, oppure il carattere CHR$ (137), muove il cursore a destra sino al punto di tabulazione più prossimo.

Esempio:

Questo esempio pone i punti di tabulazione alle colonne 15, 25 e 50 e quindi visualizza le parole uno, due e tre a partire da queste tre posizioni.

```
10 PRINT"[redacted]"
20 PRINT"[redacted]UNO[redacted]DUE[redacted]TRE"
```

TAN

TAN ritorna la tangente dell'angolo argn.

Formato:

TAN (argn)

Esempio:

```
?TAN(3.2)                    Visualizza 0.0584738547
XY(1)=TAN(180*π/180)
```

TI, TI$

TI e TI$ rappresentano due variabili di tempo del sistema.

Formato:

TI	Numero di "attimi" dall'accensione
TI$	Stringa con il tempo orario

Esempio:

```
?TI
TI$="081000"
```

Per l'impiego delle variabili TI e TI$ vi rimandiamo al capitolo 5.

USR

USR è una funzione del sistema che passa un parametro ad una routine in Assembler scritta dall'utente e riceve alla fine un parametro di ritorno. L'indirizzo di inizio della routine è contenuto nelle posizioni di memoria 1 e 2.

Formato·

USR (arg)

La funzione USR è descritta in dettaglio nel capitolo 7.

VAL

La funzione VAL ritorna l'equivalente numerico della stringa dato$.

Formato:

VAL (dato$)

Il numero ritornato con VAL può essere usato in qualunque espressione numerica.

VAL rimuove tutti gli spazi bianchi posti anteriormente alla stringa poi analizza il primo carattere. Se questo carattere non è una cifra VAL ritorna il valore 0. Se è invece una cifra VAL analizza tutti i caratteri/cifra verso destra sino ad incontrare un carattere che non sia una cifra. Il numero ritornato corrisponde quindi a questo

pacchetto di cifre. In altre parole VAL corrisponde alle cifre poste a sinistra nella stringa dato$.

Esempio:

```
A=VAL("123")
NN=VAL(B$)
```

FUNZIONI DI ELABORAZIONE DEI TESTI DEI CALCOLATORI CBM 8000

La serie CBM 8000 usufruisce di queste funzioni speciali.

BELL

La funzione BELL fa suonare il campanello del calcolatore (purchè il calcolatore abbia questa opzione).

Formato:

CHR$ (7) oppure < ESC > < RVS > g

Il campanello suona ogni volta che il suo comando viene inserito in una PRINT. Il campanello suona anche quando viene superata la colonna 75 dello schermo o quando il calcolatore viene acceso. Se sullo schermo è presente una finestra di dimensioni ridotte, allora il campanello suona quando viene superata la quinta colonna dall'estremo destro della finestra.

Esempio:

```
100 PRINT CHR$(7)
```

DELETE LINE (BASIC 4.0)

Questa funzione cancella una linea sul display e fa scorrere verso l'alto di una riga tutto il testo sottostante.

Formato:

CHR$ (21) oppure < ESC > < RVS > u

Per usare questa funzione inserite uno dei due possibili formati in una PRINT. La linea che viene cancellata è quella su cui è presente il cursore. Ricordiamo che viene cancellata la linea solo sul display, ma non in memoria. Di conseguenza questa funzione può essere usata per modificare lo schermo, ma non per cancellare dati in memoria.

Esempio:

```
PRINT"<HOME><CRSR↓><CRSR↓><CRSR↓><ESC><RVS>U"
```

Cancella la quarta linea del display

ERASE BEGIN

Questa funzione cancella la porzione sinistra di una linea dall'inizio sino al cursore.

Formato:

CHR$ (150) oppure < ESC > < RVS > V

Per usare questa funzione inserite uno dei due possibili formati in una PRINT. La linea su cui è presente il cursore sarà cancellata dall'inizio sino alla posizione stessa del cursore. Attenzione però che i dati non sono cancellati anche in memoria, per cui questa funzione può essere usata solo per l'editing dello schermo.

Esempio:

```
100 PRINT TAB(20);CHR$(150)
```

Cancella i primi 20 caratteri della linea

ERASE END

Questa funzione cancella la porzione di destra di una linea dal cursore sino alla fine della linea.

Formato:

CHR$ (22) oppure < ESC > < RVS > v

Per usare questa funzione inserite uno dei due possibili formati in una PRINT. La linea del display su cui è presente il cursore viene cancellata dalla posizione del cursore sino alla fine della linea. I dati in memoria non vengono però cancellati, per cui questa funzione può essere usata solo per l'editing dello schermo.

Esempio:

```
100 PRINT TAB(20);CHR$(22)
```

Cancella la linea a destra del carattere 20

GRAPHIC

La funzione GRAPHIC cambia la visualizzazione sullo schermo dai caratteri di testo a quelli grafici.

Formato:

CHR$ (142) oppure < ESC > < RVS > N

La funzione GRAPHIC viene abilitata inserendo uno dei due formati in una PRINT. L'insieme di caratteri standard viene selezionato per quei caratteri che hanno un simbolo grafico. Gli spazi tra le linee vengono tolti per migliorare la qualità del grafico.

Gli effetti della funzione GRAPHIC sono rimossi dalla funzione TEXT.

Esempio:

```
PRINT CHR$(142)        Seleziona la visualizzazione grafica
```

INSERT LINE

Questa funzione inserisce una linea bianca sullo schermo nella posizione del cursore.

Formato:

CHR$ (149) oppure < ESC > < RVS > m

Per ottenere l'inserimento di una linea con questa funzione dovete usare uno dei due formati in una PRINT. La linea viene inserita ove è presente il cursore. Il testo sottostante scorre verso il basso di una riga e l'ultima riga in basso esce dallo schermo.

Questa funzione modifica unicamente lo schermo, ma non il contenuto della memoria, per cui può essere usata solamente per l'editing dello schermo.

Esempio:

```
PRINT "<HOME><CRSR↓><CRSR↓><CRSR↓><ESC><RVS>M"   Inserisce una linea nella quarta
                                                 riga del display
```

SCROLL DOWN E SCROLL UP

Queste due funzioni fanno scorrere il testo sullo schermo di una linea verso il basso o verso l'alto.

Formato:

CHR$ (153) oppure < ESC > < RVS > Q SCROLL DOWN
CHR$ (25) oppure < ESC > < RVS > q SCROLL UP

Mediante le funzioni SET BOTTOM e SET TOP è possibile definire una finestra sullo schermo dei calcolatori CBM. Nell'ambito di questa finestra le funzioni SCROLL DOWN e SCROLL UP fanno scorrere il testo verso il basso o verso l'alto. SCROLL DOWN fa scorrere tutto il testo entro la finestra di una riga verso il basso. La prima riga diviene bianca, mentre l'ultima esce dalla finestra. SCROLL UP fa l'opposto facendo salire di una riga il testo nella finestra. Queste due funzioni possono essere abilitate inserendo uno dei formati in una PRINT.

SCROLL DOWN e SCROLL UP modificano lo schermo, ma non la memoria per cui possono essere usate solo per l'editing dello schermo.

Esempio:

```
10 PRINT CHR$(25)      Fa scorrere di una riga in alto nella finestra
```

SET BOTTOM E SET TOP

Queste due funzioni definiscono una finestra sullo schermo dei display CBM.

Formato:

```
CHR$ (143)        SET BOTTOM
CHR$ ( 15)        SET TOP
```

La funzione SET BOTTOM definisce l'angolo inferiore destro della finestra. La funzione SET TOP definisce l'angolo superiore sinistro della finestra. Per impostare la finestra si deve portare il cursore nelle posizioni in basso a destra e in alto a sinistra, nella lista di una PRINT, e in corrispondenza far seguire i comandi SET.

Per cancellare una finestra eseguite una PRINT con due caratteri HOME consecutivi.

Esempio:

Supponiamo di voler tracciare una finestra tra le righe 5 e 15 e tra le colonne 10 e 60. Con la seguente PRINT costruiamo la finestra.

```
10 PRINT"SSSSS";TAB(10);CHR$(15);"XXXXXXXXX";TAB(60);CHR$(143)
```

Con la seguente PRINT invece la cancelliamo:

```
100 PRINT "<HOME><HOME>"
```

TEXT

La funzione TEXT cancella gli effetti della funzione GRAPHIC. I caratteri che abbiano un simbolo grafico nell'insieme di caratteri standard sono commutati nella rappresentazione dell'insieme dei caratteri alternativi.

Formato:

CHR$ (14) oppure < ESC > < RVS > n

La funzione TEXT viene abilitata inserendo uno dei due formati in una PRINT.

Esempio:

```
100 PRINT CHR$(14)      Termina la grafica
```

APPENDICE A

CODICE DEI CARATTERI CBM

Questa appendice contiene le seguenti tavole:

- Parole-chiave del BASIC CBM (Tabella A-1)
- Codice ASCII CBM a 7 bit (Tabella A-2)
- Codice a 7 bit della memoria dello schermo (Tabella A-3)
- Codice a 8 bit degli insiemi di caratteri standard e alternativo (Tabella A-4)

Il significato delle tabelle A-1, A-2 e A-3 è di per se evidente ed inoltre il loro contenuto è già stato descritto. Anche il contenuto della tabella A-4, che riguarda gli insiemi di caratteri standard e alternativo, è già stato descritto ma riteniamo utile ripeterne i punti salienti in quanto questo argomento è caratteristico dei calcolatori CBM.

Le prime due colonne della tabella A-4 riportano gli insiemi di caratteri standard e alternativo per i calcolatori PET e CBM. Le altre tre colonne indicano il corrispondente codice ASCII ed il valore PEEK/POKE. I caratteri sono elencati secondo il valore ASCII in ordine crescente. Nel caso che manchi il valore ASCII, come per i caratteri in campo invertito, si è preso l'ordine del valore PEEK/POKE. Molti caratteri appaiono due volte perchè hanno due codici ASCII.

Insieme di caratteri standard. Questi caratteri sono abilitati quando il calcolatore PET 2001 viene acceso, oppure quando viene dato il comando POKE 59468,12 negli altri calcolatori CBM. Questo insieme di caratteri comprende le lettere maiuscole, i numeri, i caratteri grafici e i simboli speciali.

Insieme alternativo di caratteri. Questo insieme è abilitato nei calcolatori CBM quando vengono accesi. Oppure può essere ottenuto dando il comando POKE 59468,14. L'insieme alternativo comprende le lettere maiuscole e minuscole, i numeri e alcuni simboli speciali.

Codice ASCII CBM. ASCII significa "American Standard Code for Information Interchange". La Commodore Business Machines ha sviluppato una versione propria di codice ASCII per potervi includere alcuni caratteri particolari.

Nella tabella A-4 sono riportati i valori del codice ASCII sia in decimale che in esadecimale. Quando usate le funzioni ASC () e CHR$ () dovete inserire il valore decimale del codice ASCII.

I caratteri in campo inverso non hanno nessuna codifica ASCII.

PEEK/POKE. Il numero PEEK/POKE è quello che usate per forzare un carattere sullo schermo. Esso è anche il numero che ottenete di ritorno da una funzione PEEK per osservare il contenuto di una posizione di memoria.

I numeri PEEK/POKE sono riportati in ordine crescente nella tabella A-4 solo dopo i valori che hanno il codice ASCII.

Tabella A-1. Parole riservate del BASIC CBM

Codice (decimale)	Parola riservata	Codice (decimale)	Parola riservata	Codice (decimale)	Parola riservata	Codice (decimale)	Parola riservata
0	End of line	70	F	141	GOSUB	181	INT
1-31	Non usato	71	G	142	RETURN	182	ABS
32	spazio	72	H	143	REM	183	USR
33	!	73	I	144	STOP	184	FRE
34	"	74	J	145	ON	185	POS
35	#	75	K	146	WAIT	186	SQR
36	$	76	L	147	LOAD	187	RND
37	%	77	M	148	SAVE	188	LOG
38	&	78	N	149	VERIFY	189	EXP
39	'	79	O	150	DEF	190	COS
40	(	80	P	151	POKE	191	SIN
41	)	81	Q	152	PRINT#	192	TAN
42	*	82	R	153	PRINT	193	ATN
43	+	83	S	154	CONT	194	PEEK
44	,	84	T	155	LIST	195	LEN
45	—	85	U	156	CLR	196	STR$
46	.	86	V	157	CMD	197	VAL
47	/	87	W	158	SYS	198	ASC
48	0	88	X	159	OPEN	199	CHR$
49	1	89	Y	160	CLOSE	200	LEFT$
50	2	90	Z	161	GET	201	RIGHT$
51	3	91	[	162	NEW	202	MID$
52	4	92	\	163	TAB(	203	Unused
53	5	93	]	164	TO	204	CONCAT†
54	6	94	↑	165	FN	205	DOPEN†
55	7	95	←	166	SPC(	206	DCLOSE†
56	8	96-127	Non usato	167	THEN	207	RECORD†
57	9	128	END	168	NOT	208	HEADER†
58	:	129	FOR	169	STEP	209	COLLECT†
59	;	130	NEXT	170	+	210	BACKUP†
60	<	131	DATA	171	−	211	COPY†
61	=	132	INPUT#	172	*	212	APPEND†
62	>	133	INPUT	173	/	213	DSAVE†
63	?	134	DIM	174	↑	215	CATALOG†
64	@	135	READ	175	AND	216	RENAME†
65	A	136	LET	176	OR	217	SCRATCH†
66	B	137	GOTO	177	>	218	DIRECTORY†
67	C	138	RUN	178	=	219	?SYNTAX ERROR†
68	D	139	IF	179	<	220-254	Non usato
69	E	140	RESTORE	180	SGN	255	π

† Solo per il BASIC 4.0

Tabella A-2. Codice ASCII standard a 7-bit.

3	2	1	0	6 5 4	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0	0	0	0		NUL	DLE	SP	0	@	P	'	p
0	0	0	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0		STX	DC2	"	2	B	R	b	r
0	0	1	1		ETX	DC3	#	3	C	S	c	s
0	1	0	0		EOT	DC4	$	4	D	T	d	t
0	1	0	1		ENQ	NAK	%	5	E	U	e	u
0	1	1	0		ACK	SYN	&	6	F	V	f	v
0	1	1	1		BEL	ETB	`	7	G	W	g	w
1	0	0	0		BS	CAN	(	8	H	X	h	x
1	0	0	1		HT	EM	)	9	I	Y	i	y
1	0	1	0		LF	SUB	*	:	J	Z	j	z
1	0	1	1		VT	ESC	+	;	K	[	k	{
1	1	0	0		FF	FS	,	<	L	\	l	|
1	1	0	1		CR	GS	—	=	M	]	m	}
1	1	1	0		SO	RS	.	>	N	^	n	~
1	1	1	1		SI	US	/	?	O	—	o	DEL

NUL	Null	FF	Form feed	ETB	End of transmission block
SOH	Start of heading	CR	Carriage return	CAN	Cancel
STX	Start of text	SO	Shift out	EM	End of medium
ETX	End of text	SI	Shift in	SUB	Substitute
EOT	End of transmission	DLE	Data line escape	ESC	Escape
ENQ	Enquiry	DC1	Device control 1	FS	File separator
ACK	Acknowledge	DC2	Device control 2	GS	Group separator
BEL	Bell, or alarm	DC3	Device control 3	RS	Record separator
BS	Backspace	DC4	Device control 4	US	Unit separator
HT	Horizontal tabulation	NAK	Negative acknowledge	SP	Space
LF	Line feed	STN	Synchronous idle	DEL	Delete
VT	Vertical tabulation				

Tabella A-3. Codice a 7-bit della memoria dello schermo.

B 6 I 1 5 I I 4 I 3210 I	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0000 I	@	P		0	—	I		r
0001 I	A	Q	!	1	♠	●	▮	┴
0010 I	B	R	"	2	I	—	▬	┬
0011 I	C	S	#	3	—	♥		┤
0100 I	D	T	$	4	`	I	_	▌
0101 I	E	U	%	5		/	I	▐
0110 I	F	V	&	6	—	X	▒	▐
0111 I	G	W	'	7	I	O	I	
1000 I	H	X	(	8	I	♣	▄	▔
1001 I	I	Y	)	9	↖	I	▶	▄
1010 I	J	Z	*	:	↘	♦	▌	┘
1011 I	K	L	+	;	↙	+	├	▪
1100 I	L	\	,	<	└	▓	▪	▘
1101 I	M	┘	-	=	\	I	└	┘
1110 I	N	↑	.	>	/	π	┐	▘
1111 I	O	←	/	?	I	◥	▁	▚

Tabella A-4. Insiemi di caratteri standard e alternativo

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
				0	00	
				1	01	
				2	02	
STOP		STOP		3	03	
				4	04	
				5	05	
				6	06	
				7	07	
				8	08	
				9	09	
				10	0A	
				11	0B	
				12	0C	
RETURN		RETURN		13	0D	
				14	0E	
				15	0F	
				16	10	
CRSR↓		CRSR↓		17	11	
RVS		RVS		18	12	
HOME		HOME		19	13	
DELETE		DELETE		20	14	
				21	15	
				22	16	
				23	17	
				24	18	
				25	19	
				26	1A	
				27	1B	
				28	1C	
CRSR→		CRSR→		29	1D	
				30	1E	
				31	1F	
␣	␣	␣	␣	32	20	32
!		!		33	21	33
"		"		34	22	34
#		#		35	23	35
$		$		36	24	36
%		%		37	25	37
&		&		38	26	38
'		'		39	27	39
(		(		40	28	40
)		)		41	29	41
*		*		42	2A	42
+		+		43	2B	43
,		,		44	2C	44
-		-		45	2D	45
.		.		46	2E	46
/		/		47	2F	47
0		0		48	30	48
1		1		49	31	49
2		2		50	32	50
3		3		51	33	51
4		4		52	34	52
5		5		53	35	53
6		6		54	36	54
7		7		55	37	55
8		8		56	38	56
9		9		57	39	57
				58	3A	58
,		,		59	3B	59
<		<		60	3C	60
=		=		61	3D	61
>		>		62	3E	62
?		?		63	3F	63
@		@		64	40	0

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
A	a	a	A	65	41	1
B	b	b	B	66	42	2
C	c	c	C	67	43	3
D	d	d	D	68	44	4
E	e	e	E	69	45	5
F	f	f	F	70	46	6
G	g	g	G	71	47	7
H	h	h	H	72	48	8
I	i	i	I	73	49	9
J	j	j	J	74	4A	10
K	k	k	K	75	4B	11
L	l	l	L	76	4C	12
M	m	m	M	77	4D	13
N	n	n	N	78	4E	14
O	o	o	O	79	4F	15
P	p	p	P	80	50	16
Q	q	q	Q	81	51	17
R	r	r	R	82	52	18
S	s	s	S	83	53	19
T	t	t	T	84	54	20
U	u	u	U	85	55	21
V	v	v	V	86	56	22
W	w	w	W	87	57	23
X	x	x	X	88	58	24
Y	y	y	Y	89	59	25
Z	z	z	Z	90	5A	26
[	[	[	[	91	5B	27
\	\	\	\	92	5C	28
]	]	]	]	93	5D	29
↑	↑	↑	↑	94	5E	30
←	←	←	←	95	5F	31
				96	60	32
!	!	!	!	97	61	33
"	"	"	"	98	62	34
#	#	#	#	99	63	35
$	$	$	$	100	64	36
%	%	%	%	101	65	37
&	&	&	&	102	66	38
				103	67	39
(	(			104	68	40
)	)		)	105	69	41
*	*	*	*	106	6A	42
+	+	+	+	107	6B	43
,	,	,	,	108	6C	44
-	-	-	-	109	6D	45
.	.	.	.	110	6E	46
/	/	/	/	111	6F	47
0	0	0	0	112	70	48
1	1	1	1	113	71	49
2	2	2	2	114	72	50
3	3	3	3	115	73	51
4	4	4	4	116	74	52
5	5	5	5	117	75	53
6	6	6	6	118	76	54
7	7	7	7	119	77	55
8	8	8	8	120	78	56
9	9	9	9	121	79	57
				122	7A	58
,	,	,	,	123	7B	59
<	<	<	<	124	7C	60
=	=	=	=	125	7D	61
>	>	>	>	126	7E	62
?	?	?	?	127	7F	63
				128	80	64

Tabella A-4. Insiemi di caratteri standard e alternativo (continua).

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
				129	81	65
				130	82	66
RUN		RUN		131	83	67
				132	84	68
				133	85	69
				134	86	70
				135	87	71
				136	88	72
				137	89	73
				138	8A	74
				139	8B	75
				140	8C	76
Shifted RETURN		Shifted RETURN		141	8D	77
				142	8E	78
				143	8F	79
				144	90	80
CRSR↓		CRSR↓		145	91	81
RVS Off		RVS Off		146	92	82
CLR Screen		CLR Screen		147	93	83
INSERT		INSERT		148	94	84
				149	95	85
				150	96	86
				151	97	87
				152	98	88
				153	99	89
				154	9A	90
				155	9B	91
				156	9C	92
CRSR→		CRSR→		157	9D	93
				158	9E	94
				159	9F	95
Shifted b		Shifted b		160	A0	96
▌	▌	▌	▌	161	A1	97
▬	▬	▬	▬	162	A2	98
▔	▔	▔	▔	163	A3	99
_	_	_	_	164	A4	100
▏	▏	▏	▏	165	A5	101
▒	▒	▒	▒	166	A6	102
▕	▕	▕	▕	167	A7	103
▒	▒	▒	▒	168	A8	104
◤	▨	▨	◤	169	A9	105
▕	▕	▕	▕	170	AA	106
├	├	├	├	171	AB	107
▪	▪	▪	▪	172	AC	108
└	└	└	└	173	AD	109
┐	┐	┐	┐	174	AE	110
─	─	─	─	175	AF	111
┌	┌	┌	┌	176	B0	112
┴	┴	┴	┴	177	B1	113
┬	┬	┬	┬	178	B2	114
┤	┤	┤	┤	179	B3	115
│	│	│	│	180	B4	116
▌	▌	▌	▌	181	B5	117
▌	▌	▌	▌	182	B6	118
─	─	─	─	183	B7	119
─	─	─	─	184	B8	120
▬	▬	▬	▬	185	B9	121
┘	╱	╱	┘	186	BA	122
▪	▪	▪	▪	187	BB	123
▪	▪	▪	▪	188	BC	124
┘	┘	┘	┘	189	BD	125
▪	▪	▪	▪	190	BE	126
▪	▪	▪	▪	191	BF	127
─	─	─	─	192	C0	64

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
↑	A	A	↑	193	C1	65
▌	B	B	▌	194	C2	66
┌	C	C	─	195	C3	67
─	D	D	─	196	C4	68
─	E	E	─	197	C5	69
─	F	F	─	198	C6	70
▌	G	G	▌	199	C7	71
▌	H	H	▌	200	C8	72
┌	I	I	┌	201	C9	73
┌	J	J	┐	202	CA	74
┌	K	K	┌	203	CB	75
└	L	L	└	204	CC	76
	M	M	╲	205	CD	77
	N	N	╱	206	CE	78
┌	O	O	┌	207	CF	79
┐	P	P	┐	208	D0	80
●	Q	Q	●	209	D1	81
─	R	R	─	210	D2	82
♥	S	S	♥	211	D3	83
▌	T	T	▌	212	D4	84
	U	U	╱	213	D5	85
	V	V	╳	214	D6	86
○	W	W	○	215	D7	87
✚	X	X	✚	216	D8	88
▌	Y	Y	▌	217	D9	89
◆	Z	Z	◆	218	DA	90
+	+	+	+	219	DB	91
▒	▒	▒	▒	220	DC	92
▌	▌	▌	▌	221	DD	93
π	▩	▩	π	222	DE	94
◥	▨	▨	◥	223	DF	95
▚	▚	▚	▚	224	E0	96
▌	▌	▌	▌	225	E1	97
▬	▬	▬	▬	226	E2	98
▔	▔	▔	▔	227	E3	99
_	_	_	_	228	E4	100
▏	▏	▏	▏	229	E5	101
▒	▒	▒	▒	230	E6	102
▕	▕	▕	▕	231	E7	103
▒	▒	▒	▒	232	E8	104
◤	▨	▨	◤	233	E9	105
▕	▕	▕	▕	234		106
├	├	├	├	235		107
▪	▪	▪	▪	236		108
└	└	└	└	237		109
┐	┐	┐	┐	238		110
─	─	─	─	239		111
┌	┌	┌	┌	240		112
┴	┴	┴	┴	241		113
┬	┬	┬	┬	242		114
┤	┤	┤	┤	243		115
│	│	│	│	244		116
▌	▌	▌	▌	245		117
▌	▌	▌	▌	246		118
─	─	─	─	247		119
─	─	─	─	248		120
▬	▬	▬	▬	249		121
┘	╱	╱	┘	250		122
▪	▪	▪	▪	251		123
▪	▪	▪	▪	252		124
┘	┘	┘	┘	253		125
▪	▪	▪	▪	254		126
π	▩	▩	π	255		127

Tabella A-4. Insiemi di caratteri standard e alternativo (continua).

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
						128
	a	a				129
	b	b				130
	c	c				131
	d	d				132
	e	e				133
	f	f				134
	g	g				135
	h	h				136
	i	i				137
	j	j				138
	k	k				139
	l	l				140
	m	m				141
	n	n				142
	o	o				143
	p	p				144
	q	q				145
	r	r				146
	s	s				147
	t	t				148
	u	u				149
	v	v				150
	w	w				151
	x	x				152
	y	y				153
	z	z				154
						155
						156
						157
						158
						159
						160
						161
						162
						163
						164
						165
						166
						167
						168
						169
						170
						171
						172
						173
						174
						175
						176
						177
						178
						179
						180
						181
						182
						183
						184
						185
						186
						187
						188
						189
						190
						191

reverse

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
						192
						193
						194
						195
						196
						197
						198
						199
						200
						201
						202
						203
						204
						205
						206
						207
						208
						209
						210
						211
						212
						213
						214
						215
						216
						217
						218
						219
						220
						221
						222
						223
						224
						225
						226
						227
						228
						229
						230
						231
						232
						233
						234
						235
						236
						237
						238
						239
						240
						241
						242
						243
						244
						245
						246
						247
						248
						249
						250
						251
						252
						253
						254
						255

APPENDICE B

MESSAGGI DI ERRORE

I messaggi di errore possono apparire sia in risposta a qualunque comando o testo che voi battete erroneamente alla tastiera sia durante l'esecuzione di un programma. Tali messaggi possono essere generati sia dell'interprete BASIC sia dal sistema operativo;di ambedue ne diamo qui di seguito la lista completa.

Ogni volta che il BASIC CBM incontra un errore esso fa apparire un messaggio diagnostico preceduto da un punto interrogativo con il seguente formato:

? messaggio ERROR IN LINE numero

dove "messaggio" è il tipo di errore, che riportiamo qui di seguito in ordine alfabetico, e "numero" è il numero della linea di programma in cui si è incontrato l'errore (questa indicazione manca nel caso di programmi in modo immediato). **Subito dopo il messaggio il BASIC ritorna in modo immediato e dà l'avviso di READY.**

Accanto ad ogni errore riportiamo la descrizione della causa e le possibili correzioni che si potrebbero effettuare.

MESSAGGI DI ERRORE DEL BASIC

Messaggio	Causa e correzioni
BAD SUBSCRIPT	È stato fatto riferimento ad un elemento di un vettore o di una matrice non previsto nel dimensionamento. Le cause possibili sono: indice superiore a 10 per variabile non dichiarata in una DIM; indice superiore a quello dichiarato nella DIM; indice che faccia riferimento ad una "dimensionalità" diversa da quella specificata nella DIM. Riportate nei limiti consentiti l'indice oppure cambiate gli estremi nell'istruzione DIM.
CANT'CONTINUE	A seguito di un comando CONT il programma non continua forse perchè alterato o fermato da un errore. Dopo un messaggio di errore un programma non può mai continuare!

Correggete l'errore ed eventualmente date RUN. Potreste anche tentare di far ripartire il programma con un GOTO al punto di interruzione.

DIVISION BY ZERO

Avete tentato di fare una divisione con divisore zero: dividere per zero non è assolutamente permesso.

Controllate le variabili o le costanti che compongono il divisore così che esso non diventi nullo. Eventualmente ponete una istruzione di salto condizionato, se il divisore si annulla, prima di fare la divisione.

FORMULA TOO COMPLEX

Non è un errore vero e proprio, ma è una indicazione che una espressione è troppo lunga e intricata per il BASIC CBM.

Ripartite tale espressione in più parti separate e rieseguite il programma. Otterrete anche di rendere più affidabile il vostro programma.

ILLEGAL DIRECT

Avete dato in modo immediato un comando che può essere eseguito solo in modo differito. Le seguenti istruzioni possono essere eseguite solo in un programma: DATA, DEF FN, GET, GET #, INPUT e INPUT #.

Eseguite le operazioni desiderate con un piccolo programma in modo differito.

ILLEGAL QUANTITY

Avete passato ad una funzione uno o più parametri fuori dei limiti consentiti. Questo messaggio appare anche se avete richiamato la funzione USR, prima di aver memorizzato l'indirizzo della subroutine alle posizioni di memoria 1 e 2.

Controllate i limiti ammessi per la vostra funzione come indicato nel capitolo 8 e fate gli opportuni cambiamenti al programma. Per maggior sicurezza potete porre una istruzione di controllo, sui valori assunti dai parametri, prima di passarli alla funzione. Se l'errore riguarda la USR inserite le istruzioni POKE, per porre l'indirizzo della subroutine, prima di richiamarla.

NEXT WITHOUT FOR

Una istruzione NEXT non è in corrispondenza con una istruzione FOR. È possibile che abbiate dimenticato l'istruzione FOR; oppure

la variabile di ciclo, che compare in NEXT, non è la stessa che viene citata in FOR.

Controllate che il ciclo FOR NEXT, a cui eventualmente corrisponde la NEXT in errore, sia corretto.

OUT OF DATA

È stata eseguita una istruzione READ mentre le corrispondenti DATA sono già state tutte lette. Ricordatevi che per ogni variabile delle istruzioni READ deve esistere un valore corrispondente nelle istruzioni DATA.

Ampliate gli elementi della DATA oppure riducete le variabili delle READ. Potete eventualmente usare il comando RESTORE per rileggere tutte le DATA, oppure mettere in coda a quest'ultima un valore flag che quando venga incontrato avvisi che non sono più disponibili altri dati.

OUT OF MEMORY

L'utente tenta di superare la zona di memoria a lui riservata, per esempio scrivendo nuove righe di un programma. Questo messaggio può apparire anche se la zona di Stack viene superata a causa di più cicli FOR NEXT o salti GOSUB posti a fascio (nidificati) pur essendovi memoria per l'utente ancora libera.

Semplificate il programma e riducete le matrici. Se necessario frazionatelo in più programmi da eseguire separatamente.

OVERFLOW

Una operazione aritmetica ha portato ad un valore numerico superiore ai limiti consentiti; cioè ad un numero superiore a 1.70141184 E+38.

Controllate i vostri calcoli. Forse potete eliminare l'errore riducendo l'ordine di grandezza delle variabili del problema.

REDIM'D ARRAY

Il nome di una variabile con indici appare in più di una istruzione DIM. Questo errore può anche verificarsi se una variabile con indice è usata senza DIM, per cui la dimensione 10 è posta automaticamente, e poi viene successivamente dimensionata esplicitamente.

Ponete sempre le istruzioni DIM all'inizio del programma e controllate che le variabili siano dimensionate una sola volta. Non pone-

te mai la DIM nei cicli FOR NEXT o nelle subroutine.

REDO FROM START

Questo messaggio può apparire durante una istruzione INPUT, ma non costituisce un errore definitivo ("fatal"). Esso indica che avete dato una risposta di tipo diverso da quello previsto (un numero per una stringa o viceversa).

Ribattete il dato corretto. INPUT ripeterà la richiesta d'ingresso fino quando darete un dato accettabile.

RETURN WITHOUT GOSUB

Il programma incontra una istruzione RETURN senza avere prima eseguito la corrispondente GOSUB.

Inserite GOSUB o cancellate RETURN. Può anche succedere che un programma entri in una subroutine direttamente per un errore logico. In tal caso dovete rivedere il programma. Ponete sempre per precauzione una END o una STOP in testa alle subroutine.

STRING TOO LONG

Avete tentato di creare una stringa più lunga di 255 caratteri mediante dei concatenamenti (+).

Frazionate questa stringa troppo lunga e trattate separatamente le varie parti. Prima di concatenare due stringhe controllatene la lunghezza con la funzione LEN.

SYNTAX

Gli errori di sintassi si rivelano solo durante l'esecuzione di un programma. Essi sono gli errori più comuni e sono dovuti a sbagli di ortografia, o a errata interpunzione, o alla presenza di caratteri estranei, o ad uso errato di parentesi, ecc.

Esaminate attentamente la linea di programma e correggetela. Ricordatevi che gli errori di sintassi sono rilevati solo al momento dell'esecuzione per cui una linea di programma deve essere attentamente letta in precedenza.

TYPE MISMATCH

Avete tentato di attribuire una stringa ad una variabile numerica o viceversa. Oppure avete dato ad una funzione un parametro di tipo errato.

Correggete le attribuzioni di variabile errate. Se necessario consultate il capitolo 8 per l'esatta definizione dei parametri delle funzioni.

UNDEF'D STATEMENT

Avete tentato di saltare ad un numero di linea che non esiste.

Correggete l'istruzione di salto o introducete un nuovo numero di linea.

UNDEF'D FUNCTION

Avete richiamato una funzione d'utente che non era stata precedentemente definita con una istruzione DEF FN. La definizione di una funzione deve precedere il suo uso.

Definite la funzione mancante. Ponete sempre le DEF FN in testa al programma.

MESSAGGI DI ERRORE DEL SISTEMA OPERATIVO

BAD DATA

Una stringa è stata fornita in ingresso al posto di dati numerici.

Correggete i dati d'ingresso o cambiate il programma così da ricevere stringhe in ingresso.

BAD DISK

A seguito di un comando HEADER siete avvisati che il dischetto è rovinato oppure che esso manca dal drive oppure che è protetto contro la scrittura.

Controllate se il dischetto è inserito correttamente nel drive. Togliete la tacca di protezione contro la scrittura se presente. Se la superficie magnetica è rovinata allora cambiate dischetto (BASIC 4.0).

DEVICE NOT PRESENT

Nessuna periferica ha risposto alla chiamata dal Bus IEEE 488. La funzione di stato assume il valore 2 per indicare un eccessivo tempo di attesa. Questo messaggio può apparire a seguito di un qualunque comando di I/O.

Può darsi che vi sia un errore nell'identificazione della periferica per cui dovete correggere la corrispondente istruzione OPEN (oppure altre istruzoni). Se questa è corretta allora controllate che non vi sia qualche cavo scollegato oppure la periferica sia spenta o altri ancora.

FILE ALREADY EXISTS

Il nome del file da copiare mediante COPY esiste già sul dischetto di destinazione. Cancellate il vecchio file sul dischetto di destinazione prima di eseguire COPY oppure usate un altro dischetto.

FILE NOT FOUND

Il file indicato in una istruzione LOAD o OPEN non esiste.

Controllate che il dischetto o cassetta su cui lo state cercando sia quello giusto. Controllatè anche tutti i nomi dei file sul dischetto, o sulla cassetta, per assicurarvi che il file che state cercando non sia registrato con un nome parzialmente diverso.

FILE OPEN

Avete tentato di aprire un file che era già stato precedentemente aperto con un'altra OPEN.

Controllate il numero di file logico (il primo nella lista parametrica della OPEN) per essere sicuri che usate un numero diverso per ogni file. Se desiderate riaprire un file per effettuare altre operazioni di I/O, allora dovete in precedenza averlo chiuso con CLOSE.

FILE NOT OPEN

Avete tentato di accedere ad un file non aperto.

Aprite il file con OPEN.

LOAD

Questo messaggio appare in connessione con il comando LOAD (vedere il capitolo 4). È dovuto ad una cattiva registrazione precedente.

NOT INPUT FILE

Avete tentato di leggere un file su cassetta che era stato già aperto in scrittura.

Controllate che i parametri delle istruzioni READ # e OPEN siano posti correttamente per effettuare la lettura del file. In questo caso il terzo parametro della OPEN deve essere 0 (0 è anche il valore imposto per "default").

NOT OUTPUT FILE

Avete tentato di scrivere in un file su cassetta che era stato già aperto per una operazione di lettura.

Controllate che i parametri delle istruzioni PRINT # e OPEN siano posti correttamente per la scrittura del file. Il terzo parametro della OPEN deve essere 1 (o 2 se volete un EOT alla fine del file).

VERIFY ERROR

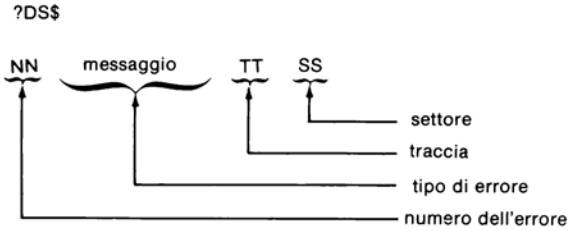
Questo messaggio viene dato a seguito del comando VERIFY (vedere il capitolo 8) ed appare quando il programma caricato in memoria non coincide con il file di programma corrispondente.

MESSAGGI DI ERRORE DEL SISTEMA DOS

COME RICHIEDERE I MESSAGGI DI ERRORE

Per leggere un messaggio di errore nell'ambito del BASIC 4.0 dovete eseguire una istruzione PRINT per visualizzare la variabile numerica DS oppure la stringa DS$.

La stringa DS$ ha il seguente formato:



Nell'ambito del BASIC < 3.0 non potete invece accedere alle variabili DS e DS$. Per esaminare lo stato di errore dovete aprire un file logico specificando l'unità fisica 8 con l'indirizzo secondario 15. Successivamente dovete richiamare quattro stringhe e visualizzarle. Ecco un esempio:

```
10 OPEN 1,8,15
20 INPUT # 1, A$, B$, C$, D$
30 PRINT A$, B$, C$, D$
40 CLOSE 1
```

A$ è il numero di errore del messaggio, B$ è il messaggio di errore, C$ è il numero della traccia e D$ il numero del settore.

Nella Tabella B-1 sono riportati numeri di traccia e di settore per tutti gli errori DOS.

Tabella B-1. Messaggi di errore del DOS.

	Numero di errore	Messaggio di errore	Traccia	Settore
Messaggi di stato	00	OK	00	00
	01	FILES SCRATCHED	# FILES	00
Errori di lettura	20	READ ERROR (Block header not found)	T	S
	21	READ ERROR (No synch character)	T	S
	22	READ ERROR (Data block not present)	T	S
	23	READ ERROR (Checksum error in data block)	T	S
	24	READ ERROR (Byte decoding error)	T	S
	27	READ ERROR (Checksum error in header)	T	S
Errori di scrittura	25	WRITE ERROR (Write-verify error)	T	S
	26	WRITE PROTECT ON	T	S
	28	WRITE ERROR (Long data block)	T	S
	29	DISK ID MISMATCH	T	S
Errori di sintassi	30	SYNTAX ERROR (General syntax)	00	00
	31	SYNTAX ERROR (Invalid command)	00	00
	32	SYNTAX ERROR (Long line)	00	00
	33	SYNTAX ERROR (Invalid file name)	00	00
	34	SYNTAX ERROR (No file given)	00	00
	39	SYNTAX ERROR (Invalid DOS command)	00	00
	50	SYNTAX ERROR (Record not present)	00	00
	51	SYNTAX ERROR (Overflow in record)	T	S
	52	SYNTAX ERROR (File too large)	T	S
Errori di file	60	WRITE FILE OPEN	00	00
	61	FILE NOT OPEN	00	00
	62	FILE NOT FOUND	00	00
	63	FILE EXISTS	00	00
	64	FILE TYPE MISMATCH	00	00
	65	NO BLOCK	T	S
	66	ILLEGAL TRACK AND SECTOR	T	S
	67	ILLEGAL SYSTEM TRACK AND SECTOR	T	S
Errori di sistema	70	NO CHANNEL	00	00
	71	DIR ERROR	00	00
	72	DISK FULL	00	00
	73	DOS MISMATCH	00	00
	74	DRIVE NOT READY	00	00

ERRORI DI LETTURA

Numero dell'errore	Messaggio	Causa dell'errore
20	Block header not found	Il controller del dischetto non riesce a trovare l'intestazione ("header") del blocco di dati richiesto. La causa può

		essere il numero di settore sbagliato oppure la perdita dell'intestazione di quel blocco.
21	No Synch character	Il controller non riesce a rilevare il segnale di sincronismo dalla traccia desiderata. La causa può essere un disallineamento della testina di lettura/scrittura, oppure la mancanza del dischetto nel drive. È possibile anche che vi sia un guasto dell'hardware.
22	Data block not present	Il controller ha tentato di leggere o verificare un blocco di dati che era stato malamente scritto in precedenza. Questo messaggio appare in seguito ad un comando BLOCK ed indica una richiesta illegale di una traccia e/o settore.
23	Checksum error in data block	Questo messaggio avvisa che vi è un errore in uno o più byte. I dati sono stati caricati nella memoria DOS e a seguito del controllo di "checksum" si è riscontrato un errore. Questi errori possono essere causati da collegamenti di terra difettosi.
24	Byte decoding error	I dati o l'intestazione "header" sono stati portati nella memoria DOS e si è creato un errore hardware a causa di una configurazione di bit, in un byte, non valida. Questi errori possono essere causati da collegamenti di terra difettosi.
27	Checksum error in header	Il controller ha rilevato un errore nella intestazione "header" del blocco di dati richiesto. Il blocco non viene trasferito nella memoria DOS. Questi errori possono essere dovuti a collegamenti di terra difettosi.

ERRORI DI SCRITTURA

Numero dell'errore	Messaggio	Causa dell'errore
25	Write verify error	Questo messaggio viene generato quando il controller rileva una diversità tra i dati

registrati e quelli contenuti nella memoria DOS.

26	Write protect on	Questo messaggio viene generato se a seguito di un comando di scrittura il controller trova l'interruttore, di protezione contro la scrittura, premuto oppure il dischetto ha la tacca di protezione coperta.
28	Long data block	Il controller tenta di rilevare il segnale di sincronismo dell'intestazione "header" successiva dopo aver scritto un blocco di dati. Se questo segnale non appare dopo un tempo prefissato il messaggio di errore viene visualizzato. L'errore è causato da una cattiva formattazione del dischetto (i dati si estendono nel blocco successivo) oppure da un difetto dell'hardware.
29	Disk id mismatch	Questo messaggio può essere generato da un dischetto che non è stato inizializzato oppure da un dischetto che ha una cattiva intestazione "header".

ERRORI DI SINTASSI

Numero dell'errore	Messaggio	Causa dell'errore
30	General syntax	Il sistema DOS non riesce ad interpretare i comandi inviati. Questo può essere causato da un numero illegale di nomi di file; per esempio due nomi di file appaiono a sinistra del segno di eguale dell'istruzione COPY.
31	Invalid command	Il DOS non riconosce i comandi. I comandi devono iniziare dalla prima posizione.
32	Long line	I comandi inviati sono più lunghi di 40 caratteri.
33	Invalid file name	Nei comandi LOAD e SAVE è stato dato un nome di file non valido.
34	No file given	Avete dimenticato il nome del file o lo avete scritto in modo non comprensibile dal DOS. Spesso ciò è dovuto alla mancanza delle virgolette (") o dei due punti (:) nella istruzione di comando.
39	Invalid dos command	Avete trasmesso un comando non riconoscibile dal DOS.

50	Record not present	Una istruzione INPUT # o GET # ha tentato di leggere un record oltre la fine di un file. Questo è sicuramente un errore se tentate di leggere un record, ma non lo è se invece vi siete posizionati in coda al file per aggiungervi altri record o un altro file.
51	Overflow in record	Una istruzione PRINT # ha tentato di scrivere più caratteri di quanto consentito in un file relativo. Ricordatevi che il carattere di ritorno del carrello conta per una posizione nel calcolo della lunghezza del record.
52	File too large	La posizione attuale del record comporterà un "overflow" del disco alla prossima operazione di scrittura su disco.

ERRORI DI FILE

Numero dell'errore	Messaggio	Causa dell'errore
60	Write file open	Questo messaggio viene generato quando tentate di aprire in lettura un file che non è stato chiuso dopo una operazione di scrittura.
61	File not open	Questo messaggio viene generato quando tentate di accedere ad un file che non è stato regolarmente aperto. Alcune volte questo messaggio non appare e il comando errato viene semplicemente ignorato.
62	File not found	Il file che si stà cercando non esiste.
63	File exists	Il file che si vuole creare esiste già sul dischetto.
64	File type Mismatch	Il tipo di file richiesto non coincide con il file presente nella directory.
65	No block	Questo messaggio può apparire a seguito del comando BLOCK-ALLOCATE. Esso indica che il blocco, che dovrebbe essere assegnato, non è più libero. In tal caso i parametri indicheranno il blocco libero, con indirizzo di traccia e settore, immediatamente superiore a quello del blocco richiesto. Se i parametri sono (00) allora tutti i blocchi con indirizzo superiore sono occupati.

66	Illegal track and sector	Avete tentato di accedere ad un settore che non esiste fisicamente. Cioè il suo indirizzo, come numeri di traccia e settore, sono oltre i valori possibili per questo dischetto. Questo messaggio lo potete riscontrare solo se lavorate con i file ad accesso diretto ("random").
67	Illegal system track and sector	Un file di dati o di programma ha tentato di accedere ad un settore del disco riservato per l'uso del sistema operativo DOS.

ERRORI DI SISTEMA

Numero dell'errore	Messaggio	Causa dell'errore
70	No channel	Il canale richiesto non è libero, oppure tutti i canali sono occupati. Nell'ambito del DOS si possono aprire al massimo cinque canali sequenziali contemporanei. I canali ad accesso diretto possono essere invece sei.
71	Dir(ectory) error	Vi è discordanza tra la mappa BAM e il il conteggio interno dei blocchi. Per correggere questo errore ri-inizializzate il dischetto per ricostituire la mappa BAM in memoria. In questo caso però i file attivi vengono interrotti.
72	Disk full	Questo messaggio viene dato in due casi: tutti i blocchi del dischetto sono stati già usati oppure non vi è più spazio nella directory (per esempio le possibili entrate della directory sono 152 per il drive CBM 2040).
73	Dos mismatch	I dati scritti su un dischetto, con una versione di DOS, possono essere letti con qualunque altra versione di DOS. La scrittura su un dischetto richiede invece che sia fatta con la stessa versione di DOS con cui il dischetto è stato inizializzato. Questo errore appare quando tentate di scrivere su un dischetto con una versione di DOS diversa da quella con cui il dischetto è stato inizializzato.
74	Drive not ready	Il drive non è pronto.

APPENDICE C

TAVOLE DI CONVERSIONE

Questa appendice contiene le seguenti tavole matematiche:

- Conversione di numeri interi dalla forma esadecimale a quella decimale.
- Potenze di due.
- Costanti matematiche.
- Potenze di sedici.
- Potenze di dieci in esadecimale.

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE

La tabella permette la conversione diretta tra numeri interi in forma esadecimale, nell'intervallo O-FFF, e numeri interi in forma decimale nell'intervallo 0-4095. Per valori superiori potete sommare gli elementi seguenti:

Esadecimale	Decimale	Esadecimale	Decimale
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Una parte frazionaria esadecimale può essere convertita in decimale, nel modo seguente:

1. Trasformate la parte frazionati in intero moltiplicandolo per 16^n, dove n è il numero di cifre significative a destra del punto. Esempio:

$$0.CA9BF3_{16} = CA9BF3_{16} \times 16^{-6}$$

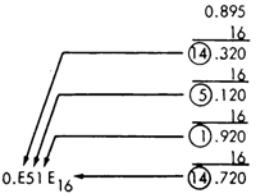
2. Trasformate questo valore in decimale intero:

$$CA9BH3_{16} = 13278195_{10}$$

3. E quindi moltiplicatelo per 16^{-6} ($16^{-6} = 596046448 \times 10^{-16}$):

$$\begin{array}{l} 13278195 \\ \times\ 596046448 \times 10^{-16} \\ \hline 0.791442096_{10} \end{array}$$

Le parti frazionarie decimali possono essere convertite in esadecimale mediante moltiplicazioni successive, della parte decimale, per 16_{10}.
Dopo ogni moltiplicazione la parte intera che ne risulta viene rimossa per costituire la parte frazionaria esadecimale. Prima è necessario però cambiare questo valore parziale in esadecimale.
Esempio: convertire 0.895_{10} in esadecimale.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

CONVERSIONE DI INTERI TRA DECIMALE ED ESADECIMALE (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

POTENZE DI DUE

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

COSTANTI MATEMATICHE

Costante	Valore decimale	Valore esadecimale
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C58F 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.29BB 075C
$\ln 10$	2.30258 40929 94046	2.4D75 3777

POTENZE DI SEDICI

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 × 10
16	1	0.62500 00000 00000 00000 × 10^{-1}
256	2	0.39062 50000 00000 00000 × 10^{-2}
4 096	3	0.24414 06250 00000 00000 × 10^{-3}
65 536	4	0.15258 78906 25000 00000 × 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 × 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 × 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 × 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 × 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807 × 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 × 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 × 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 × 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 × 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 × 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 × 10^{-18}

POTENZE DI DIECI
(in esadecimale)

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 × 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E × 16^{-2}
2710	4	0.68DB 8BAC 710C B296 × 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 × 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37 × 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858 × 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF × 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC × 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF × 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF × 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 × 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 × 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 × 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D × 16^{-12}
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE × 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 × 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 × 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 × 16^{-15}

APPENDICE D

MEMORIA ROM CON REVISIONE LIVELLO 2

In questa appendice descriviamo le differenze tra la memoria ROM con revisione livello 3, a cui si è sempre fatto riferimento in questo volume, e la memoria ROM con revisione livello 2. Passiamo in rassegna per ogni capitolo i punti diversi tra i due livelli di ROM.

Capitolo 1: ACCENSIONE DEL CALCOLATORE

Il segno di asterisco (*) sostituisce il segno (#) nella intestazione iniziale all'accensione del calcolatore:

```
*** COMMODORE BASIC ***
```

Potete far riferimento a questa diversità per capire subito quale tipo di ROM avete nel calcolatore.

Capitolo 4: VARIABILI CON INDICI

Con la ROM revisione livello 2 il massimo numero di elementi totali di un vettore o di una matrice può essere 256. Per esempio un vettore può avere al massimo 256 elementi (da 0 a 255). Una matrice a due dimensioni può avere al massimo gli indici come: (127,1) o (1,127) o (3,63) o (63,3) ecc.

Un esempio di programma, che ha questa restrizione, viene dato più oltre per la generazione dei numeri a caso.

Capitolo 5: SVILUPPO DI UN PROGRAMMA, Programmazione interattiva

Nella ROM con livello 2 di revisione la posizione di memoria che abilita il cursore a lampeggiare è la posizione 548. Per abilitare il cursore dovete usare l'istruzione:

```
80 POKE 548,0        abilitazione del cursore
                     (ROM revisione livello 2)
```

invece di:

```
80 POKE 167,0        abilitazione del cursore
                     (ROM revisione livello 3)
```

Capitolo 5: Funzione RND

RND (0) non genera numeri a caso, ma un valore quasi costante diverso per ogni calcolatore CBM.

Per avere semi a caso dovete usare −TI come abbiamo fatto nel programma che segue per la generazione di numeri a caso.

Capitolo 5: GENERAZIONE DI NUMERI A CASO

Non usate la funzione RND (−RND(0)) per generare numeri a caso, ma usate il seme −TI.

A causa della limitazione massima a 256 elementi per le variabili con indici, il programma RANDOM VERSIONE 2 non può girare con la ROM revisione livello 2. Qui di seguito vi diamo un'altra versione modificata in modo che la tabella di 1.000 elementi sia ripartita in quattro parti da 250 elementi ciascuna.

```
5 REM    VERSIONE 2A RANDOM
6 REM
10 REM   **********  P R O V A  **********
15 REM
20 REM    VISUALIZZAZIONE A CASO DI UN
30 REM    CARATTERE BATTUTO ALLA TASTIERA
35 REM
40 REM   ********************************
70 DIM T1(249),T2(249),T3(249),T4(249)
75 T=4                    :REM NUMERO DELLE TAVOLE
76 N=250                  :REM NUMERO ELEMENTI
80 GOSUB 200              :REM INIZIALIZ. TAVOLE
90 PRINT"BATTI UN TASTO O <R> PER TERMINARE";
95 N1=N:N2=N:N3=N:N4=N
100 GET C$: IF C$="" GOTO 100
105 IF C$=CHR$(13) GOTO 170
110 PRINT"□";             :REM PULISCE LO SCHERMO
120 X=RND(-TI)            :REM PONE UN NUOVO "SEME"
125 C=(ASC(C$)AND128)/2 OR (ASC(C$)AND63)
126 FOR L=1 TO 1000       :REM UNO PER OGNI PUNTO
127 T%=T*RND(1)+1
128 ON T% GOSUB 300,400,500,600
130 POKE A,C              :REM VISUALIZ. CARATTERE
140 NEXT L
160 GOTO 95
170 END
199 REM   *** SUBROUT.INIZIZLIZ.TAVOLE ***
200 FOR I=0 TO N-1:T1(I)=I    : NEXT
210 FOR I=0 TO N-1:T2(I)=I+250: NEXT
220 FOR I=0 TO N-1:T3(I)=I+500: NEXT
230 FOR I=0 TO N-1:T4(I)=I+750: NEXT
240 RETURN
299 REM   ***    SUBROUT. TAVOLA 1     ***
300 N1=N1-1
305 REM SE VUOTA VA A UN'ALTRA TAVOLA
310 IF N1<0 THEN ON INT(3*RND(1)+1) GOTO 400,500,600
320 A%=(N1+1)*RND(1)
330 A=T1(A%)+32768
340 TP=T1(A%):T1(A%)=T1(N1):T1(N1)=TP
350 RETURN
399 REM   ***    SUBROUT. TAVOLA 2     ***
400 N2=N2-1
410 IF N2<0 THEN ON INT(3*RND(1)+1) GOTO 300,500,600
420 A%=(N2+1)*RND(1)
430 A=T2(A%)+32768
```

```
440 TP=T2(A%):T2(A%)=T2(N2):T2(N2)=TP
450 RETURN
499 REM   ***    SUBROUT. TAVOLA 3     ***
500 N3=N3-1
510 IF N3<0 THEN ON INT(3*RND(1)+1) GOTO 300,400,600
520 A%=(N3+1)*RND(1)
530 A=T3(A%)+32768
540 TP=T3(A%):T3(A%)=T3(N3):T3(N3)=TP
550 RETURN
599 REM   ***    SUBROUT. TAVOLA 4     ***
600 N4=N4-1
610 IF N4<0 THEN ON INT(3*RND(1)+1) GOTO 300,400,500
620 A%=(N4+1)*RND(1)
630 A=T4(A%)+32768
640 TP=T4(A%):T4(A%)=T4(N4):T4(N4)=TP
650 RETURN
```

Capitolo 6: FILE

Questa sezione interessa solamente quegli utenti che abbiano avuto difficoltà nel leggere file di dati su cassetta usando le vecchie ROM. Se il vostro calcolatore ha la ROM con il livello 2 di revisione e dovete usare molti file, vi consigliamo allora di sostituire la ROM livello 2 con una ROM livello 3. Questo perchè la versione 3 garantisce una grande affidabilità nella lettura e scrittura dei file.

Se invece continuate a lavorare con la ROM a livello 2, allora dovrete aggiungere qualche linea di programma per ovviare agli inconvenienti di cui abbiamo detto più sopra. Quando scrivete un file di dati su nastro, la ROM di livello 2 trascura di portare il puntatore del buffer di memoria all'inizio e di lasciare un sufficiente spazio tra i record fisici sul nastro ("inter record gap")[1]. Di conseguenza quando tenterete di leggere il file potrà succedere che non riusciate a farlo correttamente e perderete quindi i dati. Ecco alcune precauzioni che potete prendere.

1. **Inizializzate all'indirizzo di partenza il puntatore del buffer della cassetta.** Siccome la ROM livello 2 non inizializza il puntatore prima di aprire il file, dovete farlo da voi mediante alcuni comandi POKE:

```
Cassetta # 1: POKE 243,122: POKE 244,2: OPEN 1,1,1
Cassetta # 2: POKE 243,58 : POKE 244,3: OPEN 2,2,1
```

Gli indirizzi di memoria 243 e 244 puntano all'indirizzo di partenza del buffer della cassetta.

2. **Forzate gli inter record gap.** La ROM livello 2 non lascia sufficiente spazio tra i record fisici ("inter record gap"). Per ovviare a questo potete forzare una maggiore spaziatura tra i record fisici mediante una subroutine che faccia avanzare il nastro dopo ogni registrazione di un record fisico.

Per sapere quando un record fisico o blocco è stato registrato basta che vi ricordiate che il buffer è sempre lungo 191 caratteri (o byte) e che appena riempito viene automaticamente registrato. Questo significa che appena aveto posto il 191-esimo caratteri nel buffer esso viene registrato in un record fisico e voi potete richiamare la subroutine per allungare l'inter record gap.

Come rilevare il riempimento del buffer

Man mano che scrivete dati sul nastro mediante una PRINT #, tenete conto della loro lunghezza e ponetela in un accumulatore. Questo accumulatore viene poi confrontato con il valore limite di 191. Quando questo valore è raggiunto la registrazione dei dati viene interrotta per inserire un inter record gap. Ecco un esempio di programma:

```
10 POKE 243,122:POKE 244,2:OPEN1,1,1
20 FOR X=1 TO 100
30 PRINT#1,X
40 A=LEN(STR$(X))+1
50 IF (QT+A)>=191 GOSUB 1000
60 QT=QT+A
70 NEXT X
80 CLOSE1
90 END
```

La linea 30 stampa una variabile. Se questa variabile è numerica (come nell'esempio) allora deve essere convertita in stringa per potere usufruire della funzione LEN che ne determina la lunghezza in caratteri (linea 40):

```
40 A = LEN (STR$(X)) + 1
```

Il valore 1 che viene aggiunto tiene conto del ritorno del carrello. Alla linea 60 vi è l'accumulatore. Alla linea 50 vi è il controllo se il valore limite di 191 viene superato per cui il buffer viene "travasato" sul nastro. In questo caso bisogna saltare alla subroutine 1000 per forzare l'inter record gap.

Avanzamento del nastro della cassetta

Le operazioni necessarie per l'avanzamento del nastro sono tre:

1. Avviare il motorino della cassetta (POKE 59411,53).
2. Porre il programma in un ciclo di attesa mentre il nastro avanza.
3. Fermare il motorino della cassetta (POKE 59411,61).

Il primo comando POKE forza il valore 53 nella posizione di memoria 59411. Tale valore 53 ha il significato di avviare il motorino di trascinamento del nastro. Il tempo di avanzamento del nastro viene stabilito con un ciclo di attesa, che durerà alcune frazioni di secondo, e di cui ne parleremo più avanti. Con il secondo comando POKE, che forza il valore 61, il motorino viene fermato.

Ecco come può essere programmato il ciclo di attesa inserito tra le due POKE:

```
1000 POKE 59411,53            :REM AVVIAMENTO DEL MOTORINO
1010 T=TI
1020 IF (TI-T)<10 GOTO 1020   :REM ATTESA PER 10 'ATTIMI'
1030 POKE 59411,61            :REM FERMA IL MOTORINO
1040 QT=0
1050 RETURN
```

Alla linea 1010 viene preso un tempo iniziale TI. TI è il tempo dell'orologio del calcolatore misurato in "attimi" (1 "attimo" = 1/60 di secondo) e può essere

azzerato sia dall'operatore che all'accensione del calcolatore PET. Alla linea 1020 si attende che passino 10 "attimi" (cioè 1/6 di secondo) con la condizione (TI-T) < 10 posta nell'istruzione IF. Durante questa attesa il nastro avanza. Quando poi il programma passa alla linea 1030 il motorino della cassetta viene fermato. Lo spazio vuoto che si è così ottenuto sul nastro si chiama "inter record gap".

In questo modo si hanno i gap tra i record tutti eguali. Se vogliamo allungarli possiamo aumentare il valore X della condizione:

$$(TI-T) < X$$

Il valore di un gap deve essere compreso tra i 5 e i 30 "attimi", ma nel dubbio è preferibile che sia superiore piuttosto che inferiore.

Attenzione però! Questa routine di attesa può non funzionare se il primo valore TI che viene dato all'orologio è vicino alle 24 ore, cioè se il calcolatore è già stato acceso da 24 ore, oppure se avete inizializzato l'orologio con un valore vicino a 24 ore. Infatti alla 24-esima ora il valore TI passa da 5184000 "attimi" a 0. È chiaro dunque che se il primo valore di TI, preso alla linea 1010, è prossimo al valore massimo di 5184000, allora la condizione TI-T < 10 sarà sempre vera e il vostro gap diverrà infinitamente lungo.

È poco probabile che questa situazione si verifichi, ma è importante che la teniate presente se lavorate in un momento in cui l'ora dell'orologio del calcolatore misurata in "attimi" è vicina al suo valore massimo.

Ecco un altro modo per programmare un ciclo di attesa:

```
POKE 59411,53          :REM AVVIAMENTO DEL MOTORINO
POKE 514,0             :REM AZZERA IL TEMPO IN 'ATTIMI'
WAIT 514,16            :REM ATTESA PER 16 'ATTIMI'
POKE 59411,61          :REM FERMA IL MOTORINO
```

Il comando POKE forza uno zero nella posizione 514. In questa posizione risiede il byte di ordine basso dell'orologio del calcolatore. Così facendo otteniamo che il valore del tempo in "attimi" sia azzerato. Il comando WAIT 514,16 inibisce il programma sino a che non siano passati 16 "attimi". Di conseguenza il nastro avanza sino a che nella posizione 514 vi sia il valore 16. Subito dopo il motorino della cassetta viene fermato.

Questo programma ha però l'inconveniente di azzerare l'orologio e di perdere quindi la possibilità di dare il tempo reale. Se avete la necessità di usare l'ora del giorno, dovete allora scegliere un altro tipo di ciclo di attesa.

Ecco un altro modo per scrivere un ciclo di attesa:

```
POKE 59411,53
FOR I=1 TO 60:NEXT I
POKE 59411,61
```

È un metodo molto semplice, ma poco preciso perchè l'esecuzione del ciclo FOR NEXT non richiede sempre lo stesso tempo. Il suo principio di funzionamento è

appunto quello di attendere che la variabile I del ciclo raggiunga il valore massimo.

Per concludere scriviamo un esempio di programma completo per la registrazione di 100 numeri su un nastro (ciclo dalla linea 20 alla 70). Ogni volta che il buffer è completo (linea 50) viene richiamata la subroutine 1000 per creare l'inter record gap.

```
10 POKE 243,122: POKE 244,2: OPEN 1,1,1
20 FOR X=1 TO 100
30 PRINT#1,X
40 A=LEN(STR$(X))+1
50 IF (QT+A)>=191 GOSUB 1000        :REM SE BUFFER PIENO CHIAMA
60 QT=QT+A                          :REM  SUBROUT. PER AVANZARE NASTRO
70 NEXT X
80 CLOSE 1
90 END
1000 POKE 59411,53                  :REM AVVIAMENTO DEL MOTORINO
1010 T=TI
1020 IF (TI-T)<10 GOTO 1020         :REM ATTESA PER 10 'ATTIMI'
1030 POKE 59411,61                  :REM FERMA IL MOTORINO
1040 QT=0                           :REM RIPRISTINA L'ACCUMULATORE
1050 RETURN
```

dove: A è la lunghezza della stringa più 1

QT è un accumulatore

Può darsi che con questi accorgimenti non abbiate più problemi nella gestione dei file, ma se dovessero permanere, allora vi raccomandiamo di installare nel vostro calcolatore una ROM livello 3.

Capitolo 7: MAPPA DI MEMORIA

Tutti i cambiamenti nel capitolo 7 sono dovuti alla diversa organizzazione della mappa di memoria tra la revisione livello 2 e quella livello 3 della ROM.

In fondo a questa appendice troverete le mappe di memoria complete corrispondenti alla varie versioni di BASIC.

La tabella D-1 descrive la ROM con revisione livello 2 impiegata nei primi calcolatori PET. La tabella D-2 riporta la ROM livello 3 usata con il BASIC 3.0. La tabella D-3 riporta la versione più recente impiegata con il BASIC 4.0.

Le tabelle D-1 e D-2 hanno lo stesso formato e riportano gli indirizzi di memoria e il loro contenuto. Ambedue questi dati sono indicati sia in forma decimale che esadecimale.

La tabella D-3 riporta invece un confronto tra la mappa del BASIC 4.0 e quella el BASIC 3.0 già vista nella tabella D-2. Nella colonna DESCRIZIONE è riportata la descrizione di ogni singola posizione come viene normalmente indicata dalla Commodore. Anche la LABEL è l'eticheta normalmente riportata nel linguaggio Assembler dalla Commodore. Le colonne BASIC 3.0 e BASIC 4.0 riportano gli indirizzi di memoria in esadecimale.

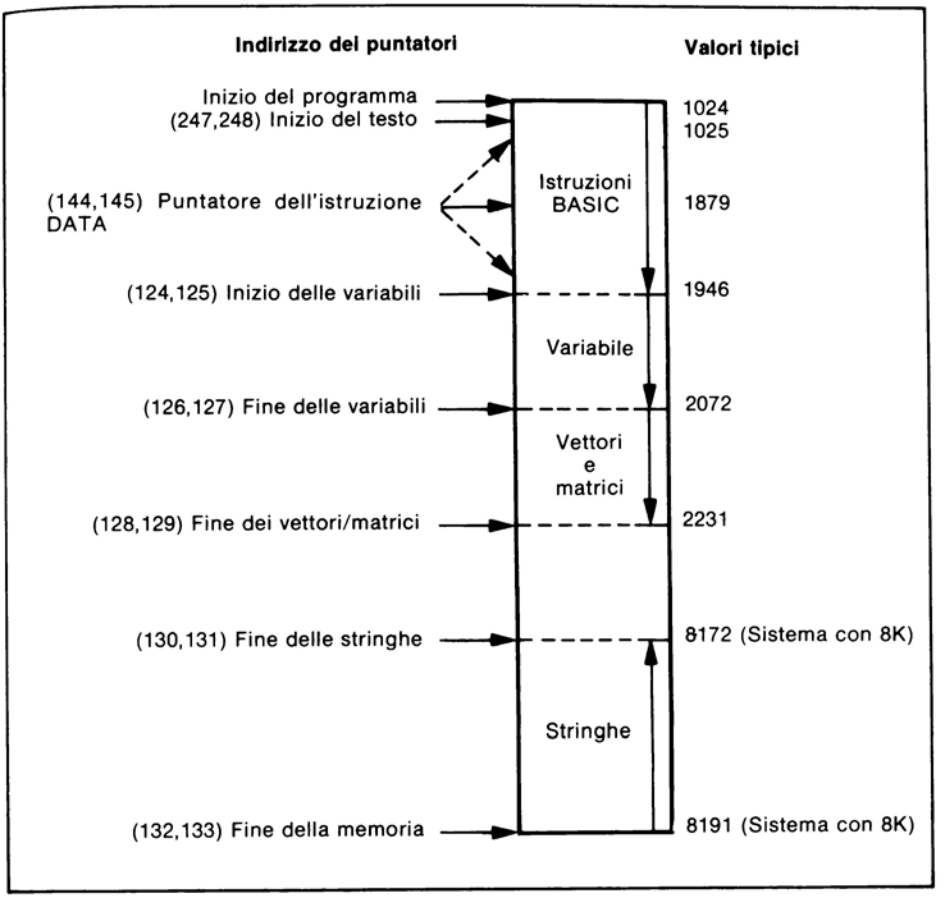


Figura D-1: Puntatori principali dell'area programmi d'utente

Per trovare un posizione in BASIC 4.0 cercate dapprima l'indirizzo in esadecimale nella tabella D-2. Ritroverete questo indirizzo esadecimale nella colonna BASIC 3.0 della tabella D-3 e confrontatelo con l'indirizzo adiacente in BASIC 4.0.

A differenza dei primi due indirizzi iniziali della tabella D-3, che rappresentano l'indirizzo di memoria 0000, tutti gli altri indirizzi 0000 identificano valori che non esistono in una delle due versioni di BASIC. Per esempio se vedete un indirizzo nella prima colonna a cui corrisponde un valore 0000 nella seconda colonna, questo significa che non esiste una posizione equivalente in BASIC 4.0. Viceversa se il valore 0000 è presente nella prima colonna e non nella seconda, significa che non esiste questa posizione in BASIC 3.0.

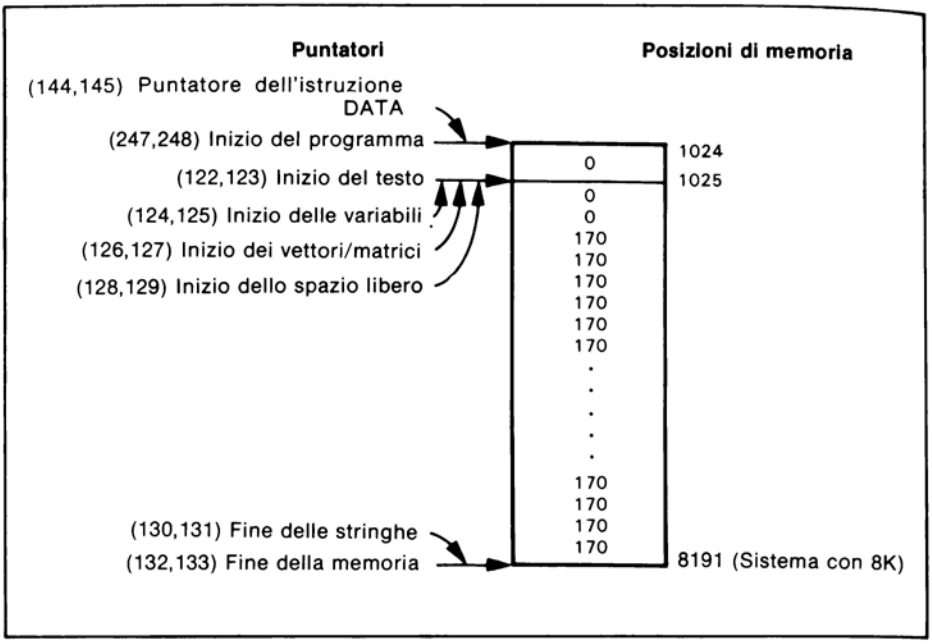


Figura D-2: Area del programma di utente al momento dell'accensione

Capitolo 7: INTERPRETE BASIC CBM

Le posizioni di memoria, che contengono i puntatori principali, sono diverse nelle due ROM livello 2 e livello 3. Tali diversità le potete constatare sostituendo la figura 7-2 con la figura D-1. Anche la figura 7-4 deve essere sostituita dalla figura D-2.

Capitolo 7: VARIABILI, formato delle variabili con virgola

Usate il seguente programma per analizzare la rappresentazione dei numeri con virgola:

```
10 INPUT A
20 X=PEEK(125)*256+PEEK(124)+2
30 PRINT A;"=";PEEK(X);PEEK(X+1);PEK(X+2);PEEK(X+3);PEEK(X+4)
40 GOTO 10
```

esso equivale a quello già dato nel capitolo 7 salvo indirizzi diversi dei comandi PEEK.

Capitolo 7: COSTANTI

Invece del puntatore (42,43) dovete porre il puntatore (124,125).

Capitolo 7: FORMATO DELLE VARIABILI CON INDICI

Usate il seguente programma per visualizzare il contenuto di una variabile con indici:

```
10 DIM A(5),B%(2,2),C$(10)          :REM ESEMPI DI ARRAY
20 FOR I=0 TO 5: A(I)=I: NEXT
30 FOR I=0 TO 2: FOR J=0 TO 2: B%(J,I)=100+3*I+J: NEXT J,I
40 FOR I=0 TO 10: C$(I)=CHR$(ASC("A")+I): NEXT
50 X=PEEK(127)*256+PEEK(126)        REM PUNTAMENTO ALL'AREA DELLE ARRAY
60 Y=PEEK(129)*256+PEEK(128)        :REM FINE DELLE ARRAY
70 FOR I=X TO Y
80 PRINT I, PEEK(I)
90 GET D$: IF D$="" THEN GOTO 90 :REM BATTERE TASTO PER ALTRO ESAME
100 NEXT
```

Esso equivale a quello già visto nel capitolo 7 salvo gli indirizzi delle PEEK alla linee 50 e 60.

Capitolo 7: PROGRAMMAZIONE IN ASSEMBLER

Nel caso di ROM livello 2 il secondo capoverso "Cima della memoria" deve essere così modificato:

2. **Fine della memoria RAM**. Le posizioni 134 e 135 contengono un puntatore alla cima della memoria che nel caso dei PET a 8K ha l'indirizzo 8192. Si può quindi spostare temporaneamente il contenuto di questo puntatore ad un valore più basso così da riservare una parte della memoria ora non più accessibile, ai vostri programmi in Assembler. Se per esempio se volete spostare in basso il puntatore di 1000 byte dovrete inserire il valore 7192 (8192-1000) convertito nei due indirizzi di ordine basso e alto:

alto basso

$$7192_{10} = 1C18_{16} \rightarrow 1C_{16} = 28_{10} \text{ e } 18_{16} = 24_{10}$$

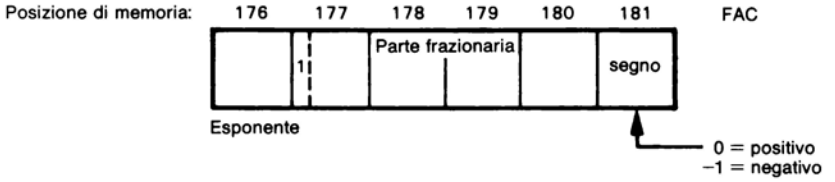
Così 24 deve essere memorizzato nella posizione 134 (byte basso) e 28 nella posizione 135 (byte alto). In BASIC potete scrivere questo programma:

```
10 AL=PEEK(134): AH=PEEK(135)    :REM SALVA ATTUALE PUNTATORE
20 POKE 132,24: POKE 135,28      :REM CIMA DELLA MEMORIA 7192
 ⋮
100 POKE 134,AL: POKE 135,AH     :REM RIPRISTINA PUNTATORE
110 END
```

Capitolo 7: USR

Nella ROM livello 2 l'accumulatore è posto in una diversa posizione di memoria rispetto alla ROM livello 3 per cui la sua descrizione, contenuta nel capitolo 7, deve essere letta nel modo seguente.

Il valore del parametro è trasferito alla subroutine tramite alcune posizioni di memoria che funzionano come un accumulatore per numeri reali (FAC) per ogni possibile funzione. L'accumulatore FAC occupa 6 byte dall'indirizzo 176 a 181 ($B0_{16}$, $B5_{16}$). Il formato del FAC è il seguente:



Come nel caso delle variabili reali, anche per questo accumulatore l'esponente è memorizzato con eccesso 128 e la parte frazionaria è normalizzata con il bit di ordine più alto, del byte 77 (cioè il byte di ordine più alto), posto eguale a 1. La differenza tra questo formato e quello delle variabili è che il bit di ordine più alto, sempre eguale a 1, in questo caso viene tenuto presente (byte 177). Un ulteriore byte (181) contiene il segno e ciò è fatto per rendere più facile la gestione dell'accumulatore.

1. PET User Notes, Volume 1, Fascicolo 6, Settembre-Ottobre 1978 pag. 14: "Cassette File Usage Summery" di Jim Butterfield.

2. Best of the Gazette, pag. 38, "On Data Files" di Michael Richter.

Tabella D-1. Mappa della memoria CBM (Rev.2)

Indirizzo di memoria		Contenuto		Descrizione
Decimale	Esadecimale	Decimale	Esadecimale	
			Page 0 (0-255)	
				USR Function Locations
0	0000	76	4C	Constant 6502 JMP instruction
1-2	0001-0002	826	033A	User address jump vector
				Terminal I/O Maintenance
3	0003	0	00	Active input device number (0=keyboard)
4	0004	0	00	No. of nulls to print after CR/LF (0=normal)
5	0005	0	00	Cursor position for POS function (0-255)
6	0006	127	7F	Terminal width (unused)
7	0007	127	7F	Limit for scanning source columns (unused)
8	0008	60	3C	Line number storage preceding buffer
9	0009	3	03	Constant
10-89	000A-0059	48	30	BASIC input line buffer (80 bytes)
90	005A	0	00	General counter for BASIC
91	005B	0	00	Delimiter flag for quote mode scan
92	005C	255	FF	Input buffer pointer, general counter
				Evaluation of Variables
93	005D	0	00	Flag for dimensioned variables
94	005E	0	00	Flag for variable type: 00=numeric FF=string
95	005F	0	00	Flag for numeric variable type: 00=floating point 80=integer
96	0060	0	00	Flag to allow reserved words in strings and remarks
97	0061	0	00	Flag to allow subscripted variable
98	0062	0	00	Flag for input type: 0=INPUT 64=GET 152=READ
99	0063	0	00	Flag sign of TAN function
100	0064	0	00	Flag to suppress output + normal - suppressed
101	0065	104	68	Index to next available descriptor
102-103	0066-0067	101	0065	Pointer to last string temporary
104-111	0068-006F	2 .	0002	Table of double-byte descriptors that point to variables (8 bytes)
112-113	0070-0071	14525	38BD	Indirect index #1
114-115	0072-0073	62983	F607	Indirect index #2
116	0074	1	01	Pseudo-register for function operands (6 bytes)
117	0075	234	EA	
118	0076	0	00	
119	0077	0	00	
120	0078	0	00	
121	0079	0	00	

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Data BASIC Storage Maintenance
122-123	007A-007B	1025	0401	Pointer to start of text
124-125	007C-007D	1946	079A	Pointer to start of variables
126-127	007E-007F	2072	0818	Pointer to end of variables
128-129	0080-0081	2231	08B7	Pointer to end of arrays
130-131	0082-0083	8192	2000	Pointer to start of strings (moving down)
132-133	0084-0085	8191	1FFF	Pointer to end of strings (top of available RAM)
134-135	0086-0087	8192	2000	Pointer to limit of BASIC memory
136-137	0088-0089	2000	07D0	Line number of current line being executed −1 in 137=direct mode statement
138-139	008A-008B	110	006E	Line number for last line executed before CONT
140-141	008C-008D	1922	0782	Pointer to next line to be executed after CONT
142-143	008E-008F	1150	047E	Line number of current DATA line
144-145	0090-0091	1879	0757	Pointer to current DATA line
146-147	0092-0093	13	000D	Next DATA item within line
148-149	0094-0095	89	0059	Current variable name
150-151	0096-0097	2032	07F0	Pointer to current variable
152-153	0098-0099	2032	07F0	Pointer to next FOR.. NEXT variable
154-155	009A-009B	31999	7CFF	Pointer to current operator in ROM table
156	009C	0	00	Mask for current logical operator
157-158	009D-009E	898	0382	Pointer to user function FN definition
159-160	009F-00A0	104	0068	Pointer to a string description
161	00A1	221	DD	Length of string
162	00A2	3	03	Constant used by garbage collection routine
163	00A3	76	4C	Constant 6502 JMP instruction
164-165	00A4-00A5	0	0000	Jump vector for user function FN
166-171	00A6-00AB	129	81	Floating point accumulator #3 (6 bytes)
172-173	00AC-00AD	0	00	Block transfer pointer #1
174-175	00AE-00AF	0	00	Block transfer pointer #2
176-181	00B0-00B5			Floating point accumulator (FAC)#1 (6 bytes)
		0	00	176 00B0 Exponent +128
		0	00	177 00B1 Fraction MSB Floating Point
		0	00	178 00B2 Fraction
		0	00	179 00B3 Fraction MSB Integer
		0	00	180 00B4 Fraction LSB
		0	00	181 00B5 Sign of fraction (0 if zero or positive, −1 if negative)
182	00B6	0	00	Copy of FAC #1 sign of fraction
183	00B7	0	00	Counter for number of bits to shift to normalize FAC #1
184-189	00B8-00BD	0	00	Floating point accumulator #2 (6 bytes)
190	00BE	0	00	Overflow byte for floating argument
191	00BF	0	00	Copy of FAC #2 sign of fraction
192-193	00C0-00C1	258	0102	Conversion pointer

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				RAM Subroutines
194-199	00C2-00C7	230	E6	Routine to fetch next BASIC character
200	00C8	173	AD	Entry to refetch current character
201-202	00C9-00CA	1929	0789	Pointer to source text
203-223	00CB-00DF	201	C9	Work area for RND function
				OS Page Zero Storage
224-225	00E0-00E1	33728	83C0	Pointer to start of line where cursor is flashing
226	00E2	0	00	Column position where cursor is flashing (0-79)
227-228	00E3-00E4	33792	8400	Utility pointer
229-230	00E5-00E6	1929	0789	End of current program
231-233	00E7-00E9	254	FE	Utility
234	00EA	0	00	Flag for quote mode 0=not quote mode
235-237	00EB-00ED	192	C0	Utility
238	00EE	0	00	No of characters in current file name
239	00EF	5	05	Current logical file number
240	00F0	255	FF	GPIB primary address
241	00F1	63	3F	GPIB device number
242	00F2	39	27	Max no of characters on current line (39,79)
243-244	00F3-00F4	634	027A	Pointer to start of current tape buffer (634 or 826)
245	00F5	23	17	Line number where cursor is flashing (0-24)
246	00F6	10	0A	I/O storage
247-248	00F7-00F8	1024	0400	OS pointer to program
249-250	00F9-00FA	3100	0C1C	Pointer to current file name
251	00FB	0	00	Number of Insert keys pushed to go
252	00FC	9	09	Serial bit shift word
253	00FD	0	00	Number of blocks remaining to read/write
254	00FE	0	09	Serial word buffer
255	00FF	243	F3	Overflow byte for binary to ASCII conversions
			Page 1 (256-511)	
256-up	0100-up	32	20	Tape read working storage (up to 511) and conversion stg. 256-318 For error correction in tape reads (62 bytes) 256-266 Binary to ASCII conversion (11 bytes)
511-down	01FF-down	0	00	Stack (down to 256)
			Page 2-3 (512-1023)	
				OS Working Storage
512-514	0200-0202	3801352	3A0108	24-hour clock incremented every 1/60 second (jiffy) Resets every 5.184.000 jiffies (24 hours) Stored in low to high order

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
515	0203	255	FF	Matrix coordinate of key depressed at current jiffy. 1-80=key 255=no key
516	0204	0	00	Status of SHIFT key: 0=unshifted (up) 1=shifted (down)
517-518	0205-0206	37916	941C	Secondary jiffy clock
519	0207	52	34	Interrupt driver flag for cassette #1 ON switch
520	0208	0	00	Interrupt driver flag for cassette #2 ON switch
521	0209	255	FF	Keyswitch PIA
522	020A	0	00	Utility
523	020B	0	00	I/O flag 0=LOAD 1=VERIFY
524	020C	0	00	I/O status byte
525	020D	0	00	Number of characters in keyboard buffer (0 to 9)
526	020E	0	00	Flag to indicate reverse field on (0=normal)
527-536	020F-0218	85	55	Keyboard buffer (10 bytes)
537-538	0219-021A	34048	8500	Hardware interrupt vector
539-540	021B-021C	0	0000	6502 BRK instruction interrupt vector
541-546	021D-0222			Input routine storage (6 bytes)
		13	0D	542 021E No. of characters on screen line
547	0223	255	FF	Key image
548	0224	1	01	Flag for cursor enable: 0=Enable 1=Disable
549	0225	11	0B	Counter to flip cursor (20 to 1)
550	0226	32	20	Copy of character at current cursor position
551	0227	0	00	Flag for cursor on/off: 0=cursor moved 1=blink started
552	0228	0	00	Flag for tape write
553-577	0229-0241			High byte of screen line addresses 553-559=128 (lines 1-7) 560-565=129 (lines 8-13) 566-572=130 (lines 14-20) 573-577=131 (lines 21-25)
578-587	0242-024B	5	05	Table of logical numbers of open files
588-597	024C-0255	5	05	Table of device numbers of open files
598-607	0256-025F	255	FF	Table of secondary address modes of open files
608	0260	0	00	Flag for input source: 0=keyboard buffer 1=screen memory
609	0261	0	00	I/O utility
610	0262	1	01	Number of open files (index into tables)

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
611	0263	0	00	Default input device number (0=keyboard)
612	0264	3	03	Default output device number (3=screen)
613	0265	0	00	Tape parity byte
614	0266	0	00	I/O utility
615	0267	0	00	I/O utility
616	0268	0	00	Byte pointer in filename transfer
617	0269	0	00	I/O utility
618	026A	255	FF	I/O utility
619	026B	0	00	I/O utility
620	026C	8	08	Serial bit count
621	026D	0	00	Count of redundant tape blocks
622	026E	0	00	Tape utility
623	026F	0	00	Cycle counter flip for each bit read from tape
624	0270	0	00	Countdown synchronization on tape write
625	0271	0	00	Tape buffer 1 index to next character
626	0272	0	00	Tape buffer 2 index to next character
627	0273	0	00	Countdown synchronization on tape read
628	0274	0	00	Flag to indicate bit/byte tape error
629	0275	0	00	Flag to indicate tape error 0=first half-byte marker not written
630	0276	0	00	Flag to indicate tape error 0=2nd half-byte marker not written /Tape dropout counter
631	0277	0	00	Tape dropout counter
632	0278	128	80	Flag for tape read current function
633	0279	9	09	Checksum utility
634-825	027A-0339	1	01	Tape buffer for cassette #1 (192 bytes)
826-1017	033A-03F9	173	AD	Tape buffer for cassette #2 (192 bytes)
1018-1023	03FA-03FF	28	1C	Utility space/unused
			Page 4-32 (1024-8191)	
1024-8191	0400-1FFF	0	00	User program area
			Page 33-128 (8192-32767)	
8192-32767	2000-7FFF	0	00	Expansion RAM
			Page 129-144 (32768-36863)	
32768-36863	8000-8FFF	12	0C	TV RAM 32768-33767 Display memory (1000 bytes)
			Page 145-192 (36864-49151)	
36864-49151	9000-BFFF	0	00	Expansion ROM
			Page 193-232 BASIC (49152-59391)	
				Pointers to BASIC Routines
49152-49153	C000-C001	50973	C71D	Pointer -1 to END*
49154-49155	C002-C003	50760	C648	Pointer -1 to FOR
49156-49157	C004-C005	52277	CC35	Pointer -1 to NEXT

* Queste posizioni di memoria contengono l'indirizzo del byte che precede la routine BASIC indicata.

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
49158-49159	C006-C007	51183	C73F	Pointer -1 to DATA
49160-49161	C008-C009	51909	CAC5	Pointer -1 to INPUT#
49162-49163	C00A-C00B	51935	CADF	Pointer -1 to INPUT
49164-49165	C00C-C00D	53104	CF70	Pointer -1 to DIM
49166-49167	C00E-C00F	52003	CB23	Pointer -1 to READ
49168-49169	C010-C011	51356	C89C	Pointer -1 to LET
49170-49171	C012-C013	51100	C79C	Pointer -1 to GOTO
49172-49173	C014-C015	51060	C774	Pointer -1 to RUN
49174-49175	C016-C017	51231	C81F	Pointer -1 to IF
49176-49177	C018-C019	50956	C70C	Pointer -1 to RESTORE
49178-49179	C01A-C01B	51071	C77F	Pointer -1 to GOSUB
49180-49181	C01C-C01D	51145	C7C9	Pointer -1 to RETURN
49182-49183	C01E-C01F	51250	C832	Pointer -1 to REM
49184-49185	C020-C021	50971	C71B	Pointer -1 to STOP
49186-49187	C022-C023	51266	C842	Pointer -1 to ON
49188-49189	C024-C025	55041	D701	Pointer -1 to WAIT
49190-49191	C026-C027	65492	FFD4	Pointer -1 to LOAD
49192-49193	C028-C029	65495	FFD7	Pointer -1 to SAVE
49194-49195	C02A-C02B	65498	FFDA	Pointer -1 to VERIFY
49196-49197	C02C-C02D	53908	D294	Pointer -1 to DEF
49198-49199	C02E-C02F	55032	D6F8	Pointer -1 to POKE
49200-49201	C030-C031	51582	C97E	Pointer -1 to PRINT#
49202-49203	C032-C033	51614	C99E	Pointer -1 to PRINT
49204-49205	C034-C035	51012	C744	Pointer -1 to CONT
49206-49207	C036-C037	50599	C5A7	Pointer -1 to LIST
49208-49209	C038-C039	51055	C76F	Pointer -1 to CLR
49210-49211	C03A-C03B	51588	C984	Pointer -1 to CMD
49212-49213	C03C-C03D	65501	FFDD	Pointer -1 to SYS
49214-49215	C03E-C03F	65471	FFBF	Pointer -1 to OPEN
49216-49217	C040-C041	65474	FFC2	Pointer -1 to CLOSE
49218-49219	C042-C043	51870	CA9E	Pointer -1 to GET
49220-49221	C044-C045	50512	C550	Pointer -1 to NEW
49222-49223	C046-C047	56075	DB0B	Pointer to SGN**
49224-49225	C048-C049	56222	DB9E	Pointer to INT
49226-49227	C04A-C04B	56106	DB2A	Pointer to ABS
49228-49229	C04C-C04D	0	0000	Pointer to USR pointer
49230-49231	C04E-C04F	53860	D264	Pointer to FRE
49232-49233	C050-C051	53893	D285	Pointer to POS
49234-49235	C052-C053	56868	DE24	Pointer to SQR
40236-49237	C054-C055	57157	DF45	Pointer to RND
49238-49239	C056-C057	55487	D8BF	Pointer to LOG
49240-49241	C058-C059	56992	DEA0	Pointer to EXP
49242-49243	C05A-C05B	57246	DF9E	Pointer to COS
49244-49245	C05C-C05D	57253	DFA5	Pointer to SIN
49246-49247	C05E-C05F	57326	DFEE	Pointer to TAN
49248-49249	C060-C061	57416	E048	Pointer to ATN
49250-49251	C062-C063	55014	D6E6	Pointer to PEEK
49252-49253	C064-C065	54868	D654	Pointer to LEN
49254-49255	C066-C067	54089	D349	Pointer to STR$
49256-49257	C068-C069	54917	D685	Pointer to VAL
49258-49259	C06A-C06B	54883	D663	Pointer to ASC
49260-49261	C06C-C06D	54724	D5C4	Pointer to CHR$
49262-49263	C06E-C06F	54744	D5D8	Pointer to LEFT$

** Queste posizioni di memoria contengono l'indirizzo del primo byte della routine BASIC indicata.

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
49264-49265	C070-C071	54788	D604	Pointer to RIGHT$
49266-49267	C072-C073	54799	D60F	Pointer to MID$
49268-57343	C074-DFFF			**BASIC Routines** **Starting Address Function** 49836 C2AC FOR NEXT stack check 49882 C2DA Insert line space marker 49949 C31D Stack overflow check 50007 C357 Error message abort 50057 C389 READY 50068 C394 Execute line 50092 C3AC Handle new line 50224 C430 Rechain lines after insert/delete 50274 C462 Input line 50297 C479 Get character from input line 50317 C48D Keyword encoder 50466 C522 Line number search 50513 C551 NEW 50586 C59A Set pointer to start of program 50600 C5A8 LIST 50761 C649 FOR NEXT 50869 C6B5 Statement processor 50930 C6F2 Statement execute 50957 C70D RESTORE 50972 C71C STOP 50974 C71E END 51013 C745 CONT 51056 C770 CLR 51061 C775 RUN 51072 C780 GOSUB 51101 C79D GOTO 51146 C7CA RETURN 51184 C7F0 DATA 51198 C7FE Next line scan 51232 C820 IF 51251 C833 REM 51267 C843 ON GOTO/GOSUB 51299 C863 Number fetch 51357 C89D LET= 51484 C91C Digit check 51583 C97F PRINT# 51589 C985 CMD 51615 C99F PRINT 51751 CA27 Print string 51780 CA44 Print character 51831 CA77 Input data error 51871 CA9F GET

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				51910 CAC6 INPUT#
				51936 CAED INPUT
				51991 CB17 Input prompt
				52004 CB24 READ
				52242 CC12 Error messages
				52278 CC36 NEXT
				52370 CC92 Format checker
				52408 CCB8 Expression evaluator
				52538 CD3A Stack argument
				52637 CD9D Symbol evaluator
				52668 CDBC Pi
				53105 CF71 DIM
				53207 CFD7 Variable table look-up
				53415 D0A7 Floating-to-integer
				53860 D264 FRE
				53880 D278 Integer-to-floating
				53893 D285 POS
				53909 D295 DEF
				54089 D349 STR$
				54724 D5C4 CHR$
				54744 D5D8 LEFT$
				54788 D604 RIGHT$
				54799 D60F MID$
				54868 D654 LEN
				54883 D663 ASC
				54917 D685 VAL
				55014 D6E6 PEEK
				55033 D6F9 POKE
				55042 D702 WAIT
				55080 D728 Subtraction
				55103 D73F Addition
				55487 D8BF LOG
				55552 D900 Multiplication
				55646 D95E Load number to AFAC
				55650 D962 Load variable to AFAC
				55780 D9E4 Division
				55924 DA74 Load Accumulator (FAC)
				55928 DA78 Load variable to FAC
				55979 DAAB Store variable from FAC
				56075 DB0B SGN
				56106 DB2A ABS
				56222 DB9E INT
				56868 DE24 SQR
				56878 DE2E Raise AFAC to power FAC
				56992 DEA0 EXP
				57157 DF45 RND
				57246 DF9E COS
				57253 DFA5 SIN
				57326 DFEE TAN

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
57344-59391	E000-E7FF			**Screen Editor** **Starting Address — Function** 57416 E048 — ATN 57525 E0B5 — Initialize BASIC system 57910 E236 — Clear screen 57981 E27D — Character fetch
58004-58986	E294-E66A			Video driver 58282 E3AA — Scroll processor 58346 E3EA — Video display routine 58185 E349 — Quote mode ($EA) switcher 58346 E3EA — Print character 58713 E559 — Scroll 1 line 58758 E586 — Interrupt Request (IRQ)
58987-59012	E66B-E684			Interrupt handler
59013-59198	E685-E73E			Clock update
59199-59227	E73F-E75B			Keyboard scan
59228-59348	E75C-E7D4			Keyboard encoding table
Page 233-240 I/O Ports and Expansion I/O (PIA s and VIA) (59392-61439)				
				Keyboard PIA (59408-59411)
59408	E810	233	E9	I/O Port A and Data Direction register
59409	E811	60	3C	Control Register A — screen blanking 52=Screen off (blanked) 60=Screen on
59410	E812	255	FF	I/O Port B and Data Direction register 255=all keys except: 254=RVS key 253=key 251=SPACE key 247= < key
59411	E813	61	3D	Control Register B — #1 cassette motor 53=motor on 61=motor off
				IEEE Port PIA (59424-59427)
59424	E820	255	FF	I/O Port A and Data Direction register PEEK (59424) reads input data.
59425	E821	188	BC	Control Register A — set output line CA2 POKE 59425,52=low POKE 59425,60=high
59426	E822	255	FF	I/O Port B and Data Direction register POKE 59426,data writes output data POKE 59426,255 before a read to Port A
59427	E823	60	3C	Control Register B — set output line CB2 POKE 59427,52=low POKE 59427,60=high

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Parallel User Port VIA (59456-59471)
59456	E840	254	FE	I/O Port B 207=#2 cassette motor on 223=#2 cassette motor off WAIT 59456,23,23 waits for vertical retrace of display Bit 1=PB1 (NFRD on IEEE connector) output line Bit 3=PB3 (ATN on IEEE connector) output line
59457	E841	255	FF	I/O Port A with handshaking
59458	E842	30	1E	Data Direction register for I/O Port B
59459	E843	0	00	Data Direction register for I/O Port A For each bit 1=output, 0=input =0 all input =255 all output
59460-59461	E844-E845	25248	62A0	(Low, high order) Read Timer 1 Counter; write to Timer 1 Latch and (high byte) initiate count
59462-59463	E846-E847	65381	FF65	(Low, high order) Read Timer 1 Latch
59464	E848	113	71	Read Timer 2 Counter low byte and reset interrupt; write to Timer 2 low byte PEEK (59464) Clock decrements every microsecond POKE 59464,n sets SR rate of shift from high (n=0) to low (n=255) for music from User Port.
59465	E849	200	C8	Read Timer 2 Counter high byte; write to Timer 2 high byte and reset interrupt. PEEK (59465) Clock decrements every millisecond
59466	E84A	1	01	Serial I/O Shift register (SR) POKE 59466,15 or 51 or 85 to generate square wave output at CB2 for playing music from User Port.
59467	E84B	0	00	Auxiliary Control register. =16 Sets SR to free-running mode for music from User Port. =0 for proper operation of tape drive
59468	E84C	14	0E	Peripheral Control register =12 for graphics on shifted characters =14 for lower-case letters on shifted characters
59469	E84D	0	00	Interrupt Flag register
59470	E84E	128	80	Interrupt Enable register
59471	E84F	255	FF	I/O Port A without handshaking
		Page 241-256 Operating System (61440-65535)		
61622-61904	F0B6-F1D0			**File Control**
61905-63532	F1D1-F82C			**Starting Address Function** 61905 F1D1 Get a character (without cursor) 61921 F1E1 Input a character (with cursor)

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				62002 F232 Display a character
				62026 F24A Close all files
				62121 F2A9 CLOSE
				62250 F32A STOP search
				62278 F346 Tape playback
				62402 F3C2 LOAD
				62481 F411 Display filename
				62515 F433 Fetch file number
				62556 F45C Number fetch
				62647 F4B7 VERIFY
				62724 F504 Fetch filename
				62741 F515 Fetch tape character
				62753 F521 OPEN
				62824 F568 Record SAVE routine
				62894 F5AE Tape header search
				62947 F5E3 Clear current tape buffer
				62957 F5ED Write tape end block
				63101 F67D Set up tape end pointer
				63108 F684 SYS
				63134 F69E SAVE
				63153 F6B1 SAVE memory block on cassette
				63273 F729 Update secondary jiffy clock
63533-64789	F82D-FD15			**Tape Control**
				63582 F85E Check for cassette on
				63615 F87F Tape read to buffer
				63684 F8C4 Write block to tape
				63765 F915 Interrupt wait
64824-65458	FD38-FFB2			**Power-On Diagnostics**
				64824 FD38 System reset SYS (64824) simulates power-on reset
				64909 FD8D Reset BASIC (does not affect User Program)
				64912 FD90 EOT-buffer compare
65472-65516	FFC0-FFEC			**Jump Vectors**
65472-65474	FFC0-FFC2	76 62753	4C F521	JMP OPEN
65475-65477	FFC3-FFC5	76 62121	4C F2A9	JMP CLOSE
65487-65489	FFCF-FFD1	76 61921	4C F1E1	JMP RDT
65490-65492	FFD2-FFD4	76 62002	4C F232	JMP WRT
65493-65495	FFD5-FFD7	76 62402	4C F3C2	JMP LOAD
65496-65498	FFD8-FFDA	76 63134	4C F69E	JMP SAVE
65499-65501	FFDB-FFDD	76 62647	4C F4B7	JMP VERIFY
65502-65504	FFDE-FFED	76 63108	4C F684	JMP SYS
65508-65510	FFE4-FFE6	76 61905	4C F1D1	JMP GETC
65514-65516	FFEA-FFEC	76 63273	4C F729	JMP Clock Update
65530-65535	FFFA-FFFF			**6502 Interrupt Vectors**
65530-65531	FFFA-FFFB	51808	CA60	Non-maskable interrupt (NMI)
65532-65533	FFFC-FFFD	64824	FD38	System reset (RESET)
65534-65535	FFFE-FFFF	58987	E66B	Interrupt request, break (IRQ+BRK)

Tabella D-2. Mappa della memoria CBM (Rev.3)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
			Page 0 (0-255)	
				USR Function Locations
0	0000	76	4C	Constant 6502 JMP instruction
1-2	0001-0002	826	033A	User address jump vector
				Evaluation of Variables and Terminal I/O Maintenance
3	0003	0	00	Search character
4	0004	0	00	Delimiter flag for quote mode scan
5	0005	255	FF	Input buffer pointer, general counter
6	0006	0	00	Flag for dimensioned variables
7	0007	0	00	Flag for variable type 00=numeric FF=string
8	0008	0	00	Flag for numeric variable type: 00=floating point 80=integer
9	0009	0	00	Flag for DATA scan, LIST quote, memory
10	000A	0	00	Flag to allow subscripted variable; FNx flag
11	000B	0	00	Flag for input type 0=INPUT 64=GET 152=READ
12	000C	0	00	Flag for ATN sign, comparison evaluation
13	000D	0	00	Flag to suppress output + normal -- suppressed
14	000E	0	00	Current I/O device for prompt-suppress
15	000F	40	28	Terminal width (unused)
16	0010	30	1E	Limit for scanning source columns (unused)
17-18	0011-0012	828	033C	Basic integer address (for SYS, GOTO, etc.)
19	0013	22	16	Index to next available descriptor
20-21	0014-0015	19	13	Pointer to last string temporary
22-29	0016-001D	2	0002	Table of double-byte descriptions that point to variables (8 bytes)
30-31	001E-001F	16451	4043	Indirect index #1
32-33	0020-0021	26119	6607	Indirect index #2
34	0022	1	01	Pseudo-register for function operands (6 bytes)
35	0023	140	8C	
36	0024	0	00	
37	0025	0	00	
38	0026	0	00	
39	0027	0	00	

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Data Storage Maintenance
40-41	0028-0029	1025	0401	Pointer to start of BASIC text
42-43	002A-002B	1920	0780	Pointer to start of variables
44-45	002C-002D	2032	07F0	Pointer to end of variables
46-47	002E-002F	2191	088F	Pointer to end of arrays
48-49	0030-0031	8192	2000	Pointer to start of strings (moving down)
50-51	0032-0033	8191	1FFF	Pointer to end of strings (top of available RAM)
52-53	0034-0035	8192	2000	Pointer to limit of BASIC memory
54-55	0036-0037	2000	07D0	Current line number Loc. 55=2 if no program yet executed
56-57	0038-0039	110	006E	Previous line number
58-59	003A-003B	1897	0769	Pointer to next line to be executed (for CONT)
60-61	003C-003D	200	00C8	Line number of current DATA line
62-63	003E-003F	1855	073F	Pointer to current DATA item
				Expression Evaluation
64-65	0040-0041	514	0202	INPUT vector
66-67	0042-0043	89	0059	Current variable name
68-69	0044-0045	2006	07D6	Pointer to current variable
70-71	0046-0047	2006	07D6	Pointer to current FOR NEXT variable
72-73	0048-0049	1279	04FF	Pointer to current operator in ROM table
74	004A	0	00	Mask for current logical operator
75-76	004B-004C	62268	F33C	Pointer to user function FN definition
77-78	0040-004E	26531	67A3	Pointer to a string description
79	004F	243	F3	Length of string
80	0050	3	03	Constant used by garbage collection routine
81	0051	76	4C	Constant 6502 JMP instruction
82-83	0052-0053	0	00	Jump vector for functions
84-89	0054-0059	211	D3	Floating point accumulator #3 (6 bytes)
90-91	005A-005B	0	0000	Block transfer pointer #1
92-93	005C-005D	0	0000	Block transfer pointer #2
94-99	005E-0063			Floating point accumulator (FAC) #1 (6 bytes)
		0	00	94 005E Exponent +128
		0	00	95 005F Fraction MSB Floating Point
		0	00	96 0060 Fraction
		0	00	97 0061 Fraction MSB Integer
		0	00	98 0062 Fraction LSB
		0	00	99 0063 Sign of fraction (0 if zero or positive. 1 if negative)
100	0064	0	00	Copy of FAC #1 sign of fraction
101	0065	•0	00	Counter for number of bits to shift to normalize FAC #1
102-107	0066-006B	0	00	Floating point accumulator #2 (6 bytes)
108	006C	0	00	Overflow byte for floating argument
109	006D	0	00	Copy of FAC #2 sign of fraction
110-111	006E-006F	258	0102	Conversion pointer

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				RAM Subroutines
112-135	0070-0087	230	E6	Routine to fetch next BASIC character
		173	AD	118 76 Entry to refetch current character
		1904	0770	119-120 77-78 Pointer into source text
136-140	0088-008C	128	80	Next random no in storage and RND work area
				OS Page Zero Storage
141-143	008D-008F	398710	061576	24-hour clock incremented every 1/60 second (jiffy) Resets every 5,184,000 jiffies (24 hours) Stored in high to low order
144-145	0090-0091	58926	E62E	Hardware interrupt vector
146-147	0092-0093	64791	FD17	6502 BRK instruction interrupt vector
148-149	0094-0095	50057	C389	NMI interrupt vector
150	0096	0	00	Status word ST (1 byte)
151	0097	255	FF	Matrix coordinate of key depressed at current jiffy 1-80=key. 255=no key
152	0098	0	00	Status of SHIFT key. 0=unshifted (up) 1=shifted (down)
153-154	0099-009A	65282	FF02	Correction factor for clock
155	009B	255	FF	Keyswitch PIA STOP and RVS flags
156	009C	0	00	Timing constant buffer
157	009D	0	00	I/O flag 0=LOAD 1=VERIFY
158	009E	0	00	Number of characters in keyboard buffer (0 to 9)
159	009F	0	00	Flag to indicate reverse field on (0=normal)
160	00A0	0	00	IEEE 488 output flag FF=character waiting
161	00A1	13	0D	Byte pointer to end of line for input
162	00A2	0	00	Utility
163-164	00A3-00A4	11, 13	0B, 0D	Cursor log (row, column)
165	00A5	63	3F	IEEE 488 output character buffer
166	00A6	255	FF	Key image
167	00A7	1	01	Flag for cursor enable. 0=Enable 1=Disable
168	00A8	17	11	Counter to flip cursor (20 to 1)
169	00A9	32	20	Copy of character at current cursor position
170	00AA	0	00	Flag for cursor on/off. 0=cursor moved 1=blink started
171	00AB	0	00	Flag for tape write
172	00AC	0	00	Flag for input source. 0=keyboard buffer 1=screen memory

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				OS Page Zero Storage (Continued)
173	00AD	0	00	I/O utility. X save flag
174	00AE	1	01	Number of open files (index into tables)
175	00AF	0	00	Default input device number (0=keyboard)
176	00B0	3	03	Default output device number (3=screen)
177	00B1	0	00	Tape parity byte
178	00B2	0	00	Flag for byte received
179	00B3	0	00	I/O utility
180	00B4	0	00	Tape buffer character
181	00B5	0	00	Byte pointer in filename transfer
182	00B6	0	00	I/O utility
183	00B7	0	00	Serial bit count
184	00B8	0	00	Tape utility
185	00B9	0	00	Cycle counter — flip for each bit read from tape
186	00BA	0	00	Countdown synchronization on tape write
187	00BB	0	00	Tape buffer 1 index to next character
188	00BC	0	00	Tape buffer 2 index to next character
189	00BD	0	00	Countdown synchronization on tape read
190	00BE	0	00	Flag to indicate bit/byte tape error
191	00BF	0	00	Flag to indicate tape error 0=first half-byte marker not written
192	00C0	0	00	Flag to indicate tape error 0=2nd half-byte marker not written
193	00C1	0	00	Tape dropout counter
194	00C2	0	00	Flag for cassette read current function 0=scan. 1-15=count. 40_{16}=load. 80_{16}=end
195	00C3	0	00	Checksum utility
196-197	00C4-00C5	33728	83CD	Pointer to start of line where cursor is flashing
198	00C6	0	00	Column position where cursor is flashing (0- 79)
199-200	00C7-00C8	33792	8400	Load start address. utility pointer
201-202	00C9-00CA	0	0000	Load end address
203-204	00CB-00CC	0	00	Tape timing constants
205	00CD	0	00	Flag for quote mode 0=not quote mode
206	00CE	0	00	Flag for tape read timer enable 0=disabled
207	00CF	0	00	Flag for EOT received from tape
208	00D0	0	00	Read character error
209	00D1	0	00	No. of characters in current file name
210	00D2	4	04	Current logical file number
211	00D3	255	FF	Current secondary address
212	00D4	4	04	Current device number
213	00D5	39	27	Current screen line length (39. 79)
214-215	00D6-00D7	0	0000	Pointer to start of current tape buffer (634 or 826)

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
216	00D8	24	18	Line number where cursor is flashing (0-24)
217	00D9	10	0A	I/O storage: last key input, buffer checksum, bit buffer
218-219	00DA-00DB	0	0000	Pointer to current file name
220	00DC	0	00	Number of Insert keys pushed to go
221	00DD	0	00	Serial bit shift word
222	00DE	0	00	Number of blocks remaining to read/write
223	00DF	0	00	Serial word buffer
224-248	00E0-00F8			High byte of screen line addresses
		128	80	224-230=128 (lines 1-7)
		129	81	231-236=129 (lines 8-13)
		130	82	237-243=130 (lines 14-20)
		131	83	244-248=131 (lines 21-25)
249	00F9	0	00	Cassette #1 status switch
250	00FA	0	00	Cassette #2 status switch
251-252	00FB-00FC	54144	D380	Tape start address
253-255	00FD-00FF	243	F3	Utility
			Page 1 (256-511)	
256-up	0100-up	32	20	Tape read working storage (up to 511) and conversion storage 256-318 For error correction in tape reads (62 bytes) 256-266 Binary to ASCII conversion (11 bytes)
511-down	01FF-down	44	2C	Stack (down to 256)
			Page 2-3 (512-1023)	
512-592	0200-0250			BASIC input line buffer (80 bytes)
		12597	3135	512-513 0200-0201 Program Counter
		50	32	514 0202 Processor status
		0	00	515 0203 Accumulator
		171	AB	516 0204 X index
		0	00	517 0205 Y index
		0	00	518 0206 Stack pointer
		15104	3B00	519-520 0207-0208 User modifiable IRQ
593-602	0251-025A	4	04	Table of logical numbers of open files
603-612	025B-0264	4	04	Table of device numbers of open files
613-622	0265-026E	255	FF	Table of secondary address modes of open files
623-632	026F-0278	3	03	Keyboard buffer (10 bytes)
633	0279	28	1C	Keyboard utility
634-825	027A-0339	28	1C	Tape buffer for cassette #1 (192 bytes)
826-1017	033A-03F9	173	AD	Tape buffer for cassette #2 (192 bytes)
1018-1019	03FA-03FB	59383	E7F7	Vector for Machine Language Monitor
1020-1023	03FC-03FF	195	C3	Utility space/unused

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				OS Page Zero Storage (Continued)
			Page 4-128 (1024-32767)	
1024-32767	0400-7FFF	0	00	User program area and Expansion RAM 4K PET 1024-4095 0400 0FFF User program area 4096-32767 1000-7FFF Expansion RAM 8K PET 1024-8191 0400-1FFF User program area 8192-32767 2000-7FFF Expansion RAM 16K PET 1024-16383 0400-3FFF User program area 16384-32767 4000-7FFF Expansion RAM 32K PET 1024-32767 0400-7FFF User program area
			Page 129-144 (32768-36863)	
32768-36863	8000-8FFF	32	20	TV RAM 32768-33767 Display memory (1000 bytes)
			Page 145-192 (36864-49151)	
36864-49151	9000-BFFF	144	90	Expansion ROM
			Page 193-232 BASIC (49152-59391)	
				Pointers to BASIC Routines
49152-49153	C000-C001	51008	C740	Pointer -1 to END*
49154-49155	C002-C003	50775	C657	Pointer -1 to FOR
49156-49157	C004-C005	52255	CC1F	Pointer -1 to NEXT
49158-49159	C006-C007	51199	C7FF	Pointer -1 to DATA
49160-49161	C008-C009	51878	CAA6	Pointer -1 to INPUT#
49162-49163	C00A-C00B	51904	CAC0	Pointer -1 to INPUT
49164-49165	C00C-C00D	53090	CF62	Pointer -1 to DIM
49166-49167	C00E-C00F	51974	CB06	Pointer -1 to READ
49168-49169	C010-C011	51372	C8AC	Pointer -1 to LET
49170-49171	C012-C013	51116	C7AC	Pointer -1 to GOTO
49172-49173	C014-C015	51076	C784	Pointer -1 to RUN
49174-49175	C016-C017	51247	C82F	Pointer -1 to IF
49176-49177	C018-C019	50991	C72F	Pointer -1 to RESTORE
49178-49179	C01A-C01B	51087	C78F	Pointer -1 to GOSUB
49180-49181	C01C-C01D	51161	C7D9	Pointer -1 to RETURN
49182-49183	C01E-C01F	51266	C842	Pointer -1 to REM
49184-49185	C020-C021	51006	C73E	Pointer -1 to STOP
49186-49187	C022-C023	51282	C852	Pointer -1 to ON
49188-49189	C024-C025	55055	D70F	Pointer -1 to WAIT
49190-49191	C026-C027	65492	FFD4	Pointer -1 to LOAD
49192-49193	C028-C029	65495	FFD7	Pointer -1 to SAVE
49194-49195	C02A-C02B	65498	FFDA	Pointer -1 to VERIFY
49196-49197	C02C-C02D	53900	D28C	Pointer -1 to DEF
49198-49199	C02E-C02F	55046	D706	Pointer -1 to POKE
49200-49201	C030-C031	51594	C98A	Pointer -1 to PRINT#

* Queste posizioni di memoria contengono l'indirizzo del byte che precede la routine BASIC indicata.

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Pointers to BASIC Routines (Continued)
49202-49203	C032-C033	51626	C9AA	Pointer -1 to PRINT
49204-49205	C034-C035	51050	C76A	Pointer -1 to CONT
49206-49207	C036-C037	50612	C5B4	Pointer -1 to LIST
49208-49209	C038-C039	50550	C576	Pointer -1 to CLR
49210-49211	C03A-C03B	51600	C990	Pointer -1 to CMD
49212-49213	C03C-C03D	65501	FFDD	Pointer -1 to SYS
49214-49215	C03E-C03F	65471	FFBF	Pointer -1 to OPEN
49216-49217	C040-C041	65474	FFC2	Pointer -1 to CLOSE
49218-49219	C042-C043	51836	CA7C	Pointer -1 to GET
49220-49221	C044-C045	50522	C55A	Pointer -1 to NEW
49222-49223	C046-C047	56133	DB45	Pointer to SGN **
49224-49225	C048-C049	56280	DBD8	Pointer to INT
49226-49227	C04A-C04B	56164	DB64	Pointer to ABS
49228-49229	C04C-C04D	0	0000	Pointer to USR pointer
49230-49231	C04E-C04F	53849	D259	Pointer to FRE
49232-49233	C050-C051	53882	D27A	Pointer to POS
49234-49235	C052-C053	56926	DE5E	Pointer to SQR
49236-49237	C054-C055	57215	DF7F	Pointer to RND
49238-49239	C056-C057	55542	D8F6	Pointer to LOG
49240-49241	C058-C059	57050	DEDA	Pointer to EXP
49242-49243	C05A-C05B	57304	DFD8	Pointer to COS
49244-49245	C05C-C05D	57311	DFDF	Pointer to SIN
49246-49247	C05E-C05F	57384	E028	Pointer to TAN
49248-49249	C060-C061	57484	E08C	Pointer to ATN
49250-49251	C062-C063	55016	D6E8	Pointer to PEEK
49252-49253	C064-C065	54870	D656	Pointer to LEN
49254-49255	C066-C067	54079	D33F	Pointer to STR$
49256-49257	C068-C069	54919	D687	Pointer to VAL
49258-49259	C06A-C06B	54885	D664	Pointer to ASC
49260-49261	C06C-C06D	54726	D5C6	Pointer to CHR$
49262-49263	C06E-C06F	54746	D5DA	Pointer to LEFT$
49264-49265	C070-C071	54790	D606	Pointer to RIGHT$
49266-49267	C072-C073	54801	D611	Pointer to MID$
49268-49297	C074-C091			Hierarchy and action addresses for operators
49298-49553	C092-C191			Table of BASIC keywords
49554-49833	C192-C2A9			BASIC error messages
				BASIC Routines
				Starting Address **Function**
49834-59343	C2AA-DFFF			49834 C2AA FOR NEXT stack check
				49880 C2D8 Insert line space marker
				49947 C31B Stack overflow check
				49960 C328 Error message abort
				50057 C389 READY
				50091 C3AB Handle new line

** Queste posizioni di memoria contengono l'indirizzo del primo byte della routine BASIC indicata.

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				BASIC Routines (Continued)
				Starting Address **Function**
				50242 C442 Rechain lines after insert/delete
				50287 C46F Input line
				50325 C495 Keyword encoder
				50476 C52C Line number search
				50523 C55B NEW
				50551 C577 CLR
				50599 C5A7 Set pointer to start of program
				50613 C5B5 LIST
				50776 C658 FOR
				50944 C700 Statement execute
				50992 C730 RESTORE
				51007 C73F STOP
				51009 C741 END
				51051 C76B CONT
				51077 C785 RUN
				51088 C790 GOSUB
				51117 C7AD GOTO
				51162 C7DA RETURN
				51200 C800 DATA
				51214 C80E Scan for next BASIC statement
				51217 C811 Scan for next BASIC line
				51248 C830 IF
				51267 C843 REM
				51283 C853 ON
				51315 C873 Number fetch
				51373 C8AD LET =
				51496 C928 Add ASCII digit to Accumulator #1
				51595 C98B PRINT#
				51601 C991 CMD
				51627 C9AB PRINT
				51740 CA1C Print string
				51769 CA39 Print character
				51791 CA4F Input data error
				51837 CA7D GET
				51879 CAA7 INPUT#
				51962 CAFA Input prompt
				51975 CB07 READ
				52220 CBFC Error messages
				52256 CC20 NEXT
				52345 CC79 Format checker
				52383 CC9F Expression evaluator
				53091 CF63 DIM
				53101 CF6D Variable table lookup
				53249 D001 Create new variable
				53420 D0AC Array table search/ create array

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				BASIC Routines (Continued)
				Starting Address **Function**
				53849 D259 FRE
				53869 D26D Integer-to-floating
				53882 D27A POS
				53888 D280 Valid direct check
				53901 D28D DEF
				54079 D33F STR$
				54726 D5C6 CHR$
				54746 D5DA LEFT$
				54790 D606 RIGHT$
				54801 D611 MID$
				54870 D656 LEN
				54885 D665 ASC
				54919 D687 VAL
				54994 D6D2 Floating-to-integer
				55016 D6E8 PEEK
				55047 D707 POKE
				55056 D710 WAIT
				55091 D733 Subtraction
				55150 D76E Addition
				55542 D8F6 LOG
				55607 D937 Multiplication
				55704 D998 Load number to AFAC
				55818 DA0A Division
				55982 DAAE Load Accumulator (FAC)
				56030 DADE Store FAC
				56072 DB08 Copy AFAC to FAC
				56088 DB18 Copy FAC to AFAC
				56133 DB45 SGN
				56164 DB64 ABS
				56280 DBD8 INT
				56526 DCCE IN line message
				56553 DCE9 Numeric-to-ASCII
				56319 DBFF String-to-floating
				56926 DE5E SQR
				56936 DE68 Power function
				57050 DEDA EXP
				57215 DF7F RND
				57304 DFD8 COS
				57311 DFDF SIN

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Screen Editor
				Starting Address **Function**
57344-5391	E000-E7FF			57384 E028 TAN
				57484 E08C ATN
				57593 E0F9 Subroutine to be moved to page 0 ($70-$87)
				57617 E111 Initial RND seed (5 bytes)
				57622 E116 Initialize BASIC system
				57897 E229 Clear screen
				57943 E257 Home cursor
				57989 E285 Character fetch
58100-58906	E2F4-E61A			Video driver
				58100 E2F4 Input from screen
				58175 E33F Quote mode ($CD) switcher
				58188 E34C Print character
				58687 E53F Scroll 1 line
58907-59113	E61B-E6E9			Interrupt Handler
59114-59127	E6EA-E6F7			Keyboard Scan
59128-59241	E6F8-E769			Keyboard Encoding Table
59242-59391	E76A-E7FF			Subroutines for Machine Language Monitor
	Page 233-240 I/O Ports and Expansion I/O (PIA's and VIA) (59392-61439)			
				Keyboard PIA (59408-59411)
59408	E810	249	F9	I/O Port A and Data Direction register
59409	E811	60	3C	Control Register A — screen blanking 52=Screen off (blanked) 60=Screen on
59410	E812	255	FF	I/O Port B and Data Direction register 255=all keys except 254=RVS key 253=¡ key 251=SPACE key 247=< key
59411	E813	61	3D	Control Registers B — #1 cassette motor 53=motor on 61=motor off
				IEEE Port PIA (59424-59427)
59424	E820	255	FF	I/O Port A and Data Direction register PEEK (59424) reads input data
59425	E821	188	BC	Control Register A — set output line CA2 POKE 59425,52=low POKE 59425,60=high
59426	E822	255	FF	I/O Port B and Data Direction registers POKE 59426, data writes output data POKE 59426,255 before a read to Port A
59427	E823	60	3C	Control Register B — set output line CB2 POKE 59427,52=low POKE 59427,60=high

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				Parallel User Port VIA (59456-59471)
59456	E840	223	DF	I/O Port B 207=#2 cassette motor on 223=#2 cassette motor off WAIT 59456,23,23 waits for vertical retrace of display Bit 1=PB1 (NFRD on IEEE connector) output line Bit 3=PB3 (ATN on IEEE connector) output line
59457	E841	255	FF	I/O Port A with handshaking
59458	E842	30	1E	Data Direction register for I/O Port B
59459	E843	0	00	Data Direction register for I/O Port A For each bit 1=output, 0=input =0 all input =255 all output
59460-59461	E844-E845	29241	7239	(Low, high order) Read Timer 1, Counter, write to Timer 1 Latch and (high byte) initiate count
59462-59463	E846-E847	65535	FFFF	(Low, high order) Read Timer 1 Latch
59464	E848	147	93	Read Timer 2 Counter low byte and reset interrupt, write to Timer 2 low byte PEEK (59464) Clock decrements every microsecond POKE 59454,n sets SR rate of shift from high (n=0) to low (n=255) for music from User Port
59465	E849	217	D9	Read Timer 2 Counter high byte, write to Timer 2 high byte and reset interrupt PEEK (59465) Clock decrements every millisecond
59466	E84A	0	00	Serial I/O Shift register (SR) POKE 59466, 15 or 85 to generate Square wave output at CB2 for playing music from User Port
59467	E84B	0	00	Auxiliary Control register =16 Sets SR to free-running mode for music from User Port =0 for proper operation of tape drive
59468	E84C	14	0E	Peripheral Control register =12 for graphics on shifted characters =14 for lower-case letters on shifted characters
59469	E84D	0	00	Interrupt Flag register
59470	E84E	128	80	Interrupt enable register
59471	E84F	255	FF	I/O Port A without handshaking
		Page 241-256 Operating System (61440-65535)		
61440-61621	F000-F0B5			Monitor messages

Tabella D-2. Mappa della memoria CBM (Rev.3) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	Esadecimale	Decimale	Esadecimale	
				GPIB Handler (IEEE 488 Bus)
				Starting Address Function
61622-61904	F0B6-F1D0			61622 F0B6 Setup for Listen, Talk, etc 61678 F0EE Send character 61736 F128 Output character immediate mode 61750 F136 Error messages 61796 F164 Send immediate Listen command, then secondary address 61807 F16F Output characters 61823 F17F Send Unlisten/ Untalk 61836 F18C Input character
				File Control
61905-63493	F1D1-F805			61905 F1D1 Get a character (without cursor) 61921 F1E1 Input a character (with cursor) 62002 F232 Output a character to any device 62062 F26E Close all files 62066 F272 Restore default I/O devices 62121 F2A9 CLOSE 62209 F301 STOP search 62223 F30F STOP key 62229 F315 Direct mode test 62402 F3C2 LOAD 62474 F40A Display filename/ fetch file number 62526 F43E Fetch LOAD/SAVE parameters 62560 F460 Fetch byte paramter 62566 F466 Send program name to GPIB 62612 F494 Tape header search 62647 F4B7 VERIFY 62670 F4CE Fetch OPEN/CLOSE parameters 62753 F521 OPEN 62886 F5A6 Find any tape header 62938 F5DA Write tape header 63036 F63C Process tape header 63108 F684 SYS 63134 F69E SAVE 63273 F729 Clock update 63344 F770 Set input device 63420 F7BC Set output device

Tabella D-1. Mappa della memoria CBM (Rev.2) (continua)

Indirizzo di memoria		Contenuto		Descizione
Decimale	**Esadecimale**	**Decimale**	**Esadecimale**	
				Tape Control
63494-64720	F806-FCD0			63494 F806 Advance tape buffer pointer 63541 F835 Gheck for cassette on 63573 F855 Tape read to buffer 63622 F886 Write block to tape 63716 F8E6 Interrupt wait
				Power-on Diagnostics
64721-64784	FCD1-FD10			64721 FCD1 System reset SYS(64721) simulates power-on reset 64766 FCFE NMI interrupt entry point
64785-65471	FD11-FFBF			64769 FD01 Table of interrupt vectors
				Machine Language Monitor
				Jump Vectors
65472-65474	FFC0-FFC2	76 62753	4C F521	JMP OPEN
65475-65477	FFC3-FFC5	76 62121	4C F2A9	JMP CLOSE
65478-65480	FFC6-FFC8	76 63344	4C F770	JMP Set Input Device
65481-65483	FFC9-FFCB	76 63420	4C F7BC	JMP Set Output Device
65484-65486	FFCC-FFCE	76 62066	4C F272	JMP Restore Default I/O Devices
65487-65489	FFCF-FFD1	76 61921	4C F1E1	JMP Input Character – RDT
65490-65492	FFD2-FFD4	76 62002	4C F232	JMP Output Character – WRT
65493-65495	FFD5-FFD7	76 62402	4C F3C2	JMP LOAD
65496-65498	FFD8-FFDA	76 63134	4C F69E	JMP SAVE
65499-65501	FFDB-FFDD	76 62647	4C F4B7	JMP VERIFY
65502-65504	FFDE-FFED	76 63108	4C F684	JMP SYS
65505-65507	FFE1-FFE3	76 62223	4C F30F	JMP Test STOP Key
65508-65510	FFE4-FFE6	76 61905	4C F1D1	JMP Get Character
65511-65513	FFE7-FFE9	76 62062	4C F26E	JMP Close all files
65514-65516	FFEA-FFEC	76 63273	4C F729	JMP Clock Update
				6502 Interrupt Vectors
65530-65531	FFFA-FFFB	65766	FCFE	Non-maskable interrupt (NMI)
65532-65533	FFFC-FFFD	64721	FCD1	System reset (RESET)
65534-65535	FFFE-FFFF	58907	E61B	Interrupt request break (IRQ+BRK)

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
0000	0000	USRPOK	$4C CONSTANT AND ADDRESS TO DISPATCH USR
0000	0000	ERRNF	ERROR CALL VALUE - ECV - NEXT WITHOUT FOR
0001	0001	ADDPRC	X
0002	0002	BUFPAG	INPUT BUFFER AT $0200
0002	0002	ADDPR2	X
0003	0003	STRSIZ	NUMBER OF LOCS PER STRING DESCRIPTOR
0003	0003	INTEGR	ONE-BYTE INTEGER FROM "@INT"
0003	0003	CHARAC	STARTING DELIMITER
0004	0004	ENDCHR	ENDING DELIMITER
0004	0004	ADDPR4	X
0005	0005	COUNT	GENERAL COUNTER FOR BASIC
0006	0006	DIMFLG	FLAG TO REMEMBER DIMENSIONED VARIABLES
0007	0007	VALTYP	FLAG FOR VARIABLE TYPE 0-NUMERIC $FF-STRING
0008	0008	INTFLG	FLAG FOR INTEGER TYPE
0008	0008	ADDPR8	X
0009	0009	GARBFL	X
0009	0009	DORES	FLAG WHETHER CAN OR CAN'T CRUNCH RESERVED WORDS
000A	000A	CLMWID	SIZE OF PRINT WINDOW
000A	000A	SUBFLG	FLAG WHICH ALLOWS SUBSCRIPTS IN SYNTAX
000B	000B	INPFLG	FLAGS INPUT OR READ
000C	000C	DOMASK	MASK USED BY RELATION OPERATIONS
000C	000C	TANSGN	FLAG SIGN OF TANGENT
0000	000D	DSDESC	DS$ LENGTH AND POINTER TO DS$
000E	C010	CHANNL	ACTIVE I/O CHANNEL #
0010	0010	ERRSN	ERROR CALL VALUE - ECV - SYNTAX
0011	0011	POKER	HOLDS ADDRESS FORE POKE COMMAND
0011	0011	LINNUM	LINE NUMBER STORAGE
0012	0012	FORSIZ	AMOUNT OF BYTES USED ON STACK FOR-NEXT
0013	0013	TEMPPT	INDEX TO NEXT AVAILABLE DESCRIPTOR
0014	0014	LASTPT	POINTER TO LAST STRING TEMP LO:HI
0016	0016	TEMPST	STORAGE FOR NUMTMP TEMP DESCRIPTORS
0016	0016	ERRRG	ECV - RETURN WITHOUT GOSUB
0017	0017	NUMLEV	NUMBER OF GOSUB LEVELS ALLOWED
001E	001E	NCMPOS	X
001F	001F	INDEX	INDRIECT INDEX #1
001F	001F	INDEX1	SAME
0021	0021	INDEX2	INDIRECT INDEX #2
0023	0023	RESHO	RES -REGISTER
0024	0024	RESMOH	[
0025	0025	ADDEND	TEMP USED BY "UMULT"
0025	0025	RESMO	[
0026	0026	RESLO	[
0028	0028	LINLEN	LENGTH OF SCREEN LINE 40-COL EDITORS
0028	0028	TXTTAB	POINTER TO START OF BASIC TEXT AREA
002A	002A	VARTAB	POINTER TO START OF VARIABLES
002A	002A	ERROD	ECV - OUT OF DATA
002C	002C	ARYTAB	POINTER TO START OF ARRAY TABLE
002E	002E	STREND	POINTER TO END OF VARIABLES
0030	0030	FRETOP	POINTER TO START OF REAL STRINGS
0032	0032	FRESPC	POINTER TO TOP OF FREE STRING SPACE
0034	0034	MEMSIZ	HIGHEST RAM ADDR AVALIBLE FOR BASIC
0035	0035	ERRFC	ECV - ILLEGAL QUANTITY
0036	0036	CURLIN	CURRENT LINE BEING EXECUTED
0038	0038	OLDLIN	LAST LINE EXECUTED (FOR CONT COMMAND)
003A	003A	OLDTXT	OLD TXTPTR (FOR CONT COMMAND) AND TEMP STORAGE
003C	003C	DATLIN	DATA LINE # FOR ERRORS

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
003E	003E	DATPTR	DATA STATEMENT POINTER
0040	0040	INPPTR	SOURCE OF INPUT ADDRESS
0042	0042	VARNAM	CURRENT VARIABLE NAME
0044	0044	FDECPT	POINTER INTO POWERS OF TEN FOR FOUT
0044	0044	VARPNT	POINTER TO VARIABLE IN MEMORY
0045	0045	ERROV	ECV - OVERFLOW
0046	0046	LSTPNT	PNTR TO LIST STRING
0046	0046	ANDMSK	THEN MASK USED BY WAIT FOR ANDING
0046	0046	FORPNT	POINTER TO CURRENT FOR-NEXT VARIABLE REFERENCE
0047	0047	EORMSK	THE MASK FOR EORING IN WAIT
0048	0048	VARTXT	POINTER INTO LIST OF VARIABLES
0048	0048	OPPTR	POINTER TO CURRENT OPERATOR IN TABLE
004A	004A	OPMASK	MASK CREATED BY CURRENT OPERATOR
004B	004B	GRBPNT	POINTER USED IN GARBAGE COLLECTION
004B	004B	TEMPF3	A THIRD FAC TEMPORARY 4-BYTES
004B	004B	DEFPNT	POINTER USED IN FUNCTION DEFINITION
004D	004D	DSCPNT	POINTER TO A STRING DESCRIPTION
004D	004D	ERROM	ECV - OUT OF MEMORY
0050	0050	FOUR6	VARIABLE CONSTANT USED BY GARB COLLECT
0051	0051	BUFLEN	INPUT BUFFER MAX SIZE+1
0051	0051	JMPER	$4C CONSTANT AND ADDRESS USED TO DISPATCH FUNCS
0052	0052	SIZE	X
0053	0053	OLDOV	THE OLD OVERFLOW
0054	0054	TEMPF1	A FAC TEMP 4-BYTES
0055	0055	ARYPNT	A POINTER USED IN ARRAY BUILDING
0055	0055	HIGHDS	DESTINATION OF HIGHEST ELMENT IN BLT.
0057	0057	HIGHTR	SOURCE OF HIGHEST ELEMENT TO MOVE
0059	0059	TEMPF2	A FAC TEMP 4-BYTES
005A	005A	DECCNT	NUMBER OF PLACES BEFORE DECIMAL POINT
005A	005A	LOWDS	LOCATION OF LAST BYTE TRANSFERRED INTO
005A	005A	ERRUS	ECV - UNDEF'D STATEMENT
005B	005B	TENEXP	BASE TEN EXPONENT FOR FIN AND FOUT
005C	005C	GRBTOP	A POINTER USED IN GARBAGE COLLECTION
005C	005C	DPTFLG	FLAG IF A DECIMAL POINT HAS BEEN INPUT
005C	005C	LOWTR	LAST THING TO MOVE IN BLT.
005D	005D	EXPSGN	SIGN OF BASE TEN EXPONENT
005D	005D	EPSGN	X
005E	005E	DSCTMP	THIS IS WHERE TEMP DESCS ARE BUILT
005E	005E	FAC	THE MAIN FLOATING POINT ACCUMULATOR
005E	005E	FACEXP	THE EXPONENT BYTE
005F	005F	FACHO	[MOST SIGNIFICANT BYTE OF MANTISSA
0060	0060	FACMOH	[ONE MORE
0061	0061	INDICE	INDICE IS SET UP HERE BY "@INT"
0061	0061	FACMO	[MIDDLE ORDER OF MANTISSA
0062	0062	FACLO	[LEAST SIG BYTE OF MANTISSA
0063	0063	FACSGN	SIGN OF FAC (0 OR -1) WHEN UNPACKED
0064	0064	DEGREE	A CONT USED BY POLYNOMIALS
0064	0064	SGNFLG	SIGN OF FAC IS PRESERVED HERE BY FIN
0065	0065	BITS	COUNTER FOR # OF BIT SHIFTS TO NORMALIZE FAC
0066	0066	ARGEXP	THE ARG REGISTER EXPONENT
0067	0067	ARGHO	[
0068	0068	ARGMOH	[
0069	0069	ARGMO	[
006A	006A	ARGLO	[
006B	006B	ARGSGN	THE SIGN (SAME AS FAC)
006B	006B	ERRBS	ECV - BAD SUBSCRIPT

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
006C	006C	STRNG1	POINTER TO A STRING OR DESCRIPTOR
006C	006C	ARISGN	A SIGN REFLECTING THE RESULT
006D	006D	FACOV	OVERFLOW BYTE OF THE FAC
006E	006E	BUFPTR	POINTER TO BUF USED BY "CRUNCH ROUTINE"
006E	006E	STRNG2	POINTER TO STRING OR DESC.
006E	006E	POLYPT	POINTER INTO POLYNOMIAL COEFFICIENTS.
006E	006E	CURTOL	ABSOLUTE LINEAR INDEX IS FORMED HERE
006E	006E	FBUFPT	POINTER INTO FBUFFER USED IN FOUT.
0070	0070	CHRGET	ROUTINE - GETS NEXT CHARACTER FROM BASIC TEXT
0076	0076	CHRGOT	ROUTINE -REGETS CURRENT CHARACTER FROM BASIC TEXT
0077	0077	TXTPTR	POINTER TO CURRENT SOURCE TEXT
0078	0078	ERRDD	ECV - REDIM'D ARRAY
007D	007D	ØNUM	LABEL IN CHRGET
0080	0080	ENDTK	TOKEN - END
0081	0081	FORTK	TOKEN - FOR
0083	0083	DATATK	TOKEN - DATA
0085	0085	ERRDVO	ECV - DIVISION BY ZERO
0087	0087	CHRRTS	LABEL IN CHRGET
0088	0088	RNDX	NEXT RANDOM NUMBER - INITIAL LOAD FROM ROM
0089	0089	GOTOTK	TOKEN - GOTO
008B	008B	ZZ7	X
008D	008D	CTIMR	24 HR CLOCK 1/60 OF SEC
008D	008D	GOSUTK	TOKEN - GOSUB
008F	008F	REMTK	TOKEN - REM
0095	0095	ERRID	ECV - ILLEGAL DIRECT
0096	0096	CSTAT	I/O OPERATION STATUS BYTE (VARIABLE ST)
0099	0099	PRINTK	TOKEN - PRINT
00A2	00A2	SCRATK	TOKEN - NEW
00A3	00A3	TABTK	TOKEN - TAB
00A3	00A3	ERRTM	ECV - TYPE MISMATCH
00A4	00A4	TOTK	TOKEN - TO
00A5	00A5	FNTK	TOKEN - FN
00A6	00A6	SPCTK	TOKEN - SPC
00A7	00A7	THENTK	TOKEN - THEN
00A8	00A8	NOTTK	TOKEN - NOT
00A9	00A9	STEPTK	TOKEN - STEP
00AA	00AA	PLUSTK	TOKEN - +
00AB	00AB	MINUTK	TOKEN - -
00B0	00B0	ERRLS	ECV - STRING TO LONG
00B1	00B1	GREATK	TOKEN - >
00B2	00B2	EQULTK	TOKEN - =
00B3	00B3	LESSTK	TOKEN - <
00B4	00B4	ONEFUN	TOKEN - SGN START OF SINGLE PARM FUNCTIONS
00BF	00BF	ERRBD	ECV - FILE DATA
00C6	00C6	TRMPOS	X
00C7	00C7	LASNUM	TOKEN - CHR$ LAST FUNC WITH ARITHMETIC PARMS
00C8	00C8	ERRST	ECV - FORMULA TOO COMPLEX
00CB	00CB	GOTK	TOKEN - GO (GO TO)
00DB	00DB	ERRCN	ECV - CAN'T CONTINUE
00E9	00E9	ERRUF	ECV - UNDEF'D FUNCTION
00FF	00FF	PI	VALUE OF PI SYMBOL
00FF	00FF	LOFBUF	START OF FOUT STRING FOR STRD AND TI$
0100	0100	FBUFFR	FOUT BUFFER HOLDS ASCII STRING FOR OUTPUT
01FB	01FB	STKEND	TOP OF STACK FOR BASIC
01FF	01FF	ZZ1	X
01FF	01FF	ZZ5	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
01FF	01FF	ZZ4	X
0200	0200	BUF	BASIC INPUT BUFFER (80 CHARACTERS-BYTES LONG)
0200	0200	BUFOFS	SAME AS ABOVE
0201	0201	ZZ2	X
0202	0202	ZZ3	X
0400	0400	RAMLOC	BEGINING OF RAM AVALIABLE FOR BASIC TEXT
0000	8000	OFFSET	*VALUE USED IN ASSEMBLY - ROM VERSION
0000	8B86	ZZ8	X
C000	B000	ROMLOC	BEGINING OF BASIC ROMS -V2=$C000 V4=$B000
C000	B000	STMDSP	START OF COMMAND DISPATCH TABLE
C046	B066	FUNDSP	START OF FUNCTION DISPATCH TABLE
C04C	B06C	USRLOC	X
C074	B094	OPTAB	START OF MATH OPERATORS DISPATCH TABLE
C089	B0A9	NEGTAB	UNITARY NEGATE DISPATCH (.BYTE 125,DISPATCH)
C08C	B0AC	NOTTAB	NOT OPERATOR DISPATCH (.BYTE 90,DISPATCH)
C08F	B0AF	PTDORL	COMPARISON DISPATCH (.BYTE 100,DISPATCH)
C092	B0B2	RESLST	START OF RESERVED WORD LIST (ASCII,END(OR $80))
C192	B20D	ERRTAB	START OF BASIC ERROR MESSAGE STORAGE
C28B	B306	ERR	MESSAGE - "ERROR"
C292	B30D	INTXT	MESSAGE - "IN"
C297	B312	REDDY	MESSAGE - "READY"
C2A2	B31B	BRKTXT	MESSAGE - "BREAK"
C2AA	B322	FNDFOR	PEEKS AT THE STACK FOR AN ACRTIVE "FOR" LOOP
C2AF	B327	FFLOOP	X
C2C4	B33C	CMPFOR	X
C2D0	B348	ADDFRS	X
C2D7	B34F	FFRTS	X
C2D8	B350	BLTU	"OPENS UP" A SPACE IN BASIC FOR A NEW LINE
C2DF	B357	BLTUC	X
C2FC	B374	BLT1	X
C308	B380	BLTLP	X
C30C	B384	MOREN1	X
C313	B38B	DECBLT	X
C31B	B393	GETSTK	TEST FOR STACK-TOO-DEEP ERROR
C328	B3A0	REASON	CHECKS FOR AVALIABLE MEMORY SPACE
C332	B3AA	TRYMOR	X
C336	B3AE	REASAV	X
C341	B3B9	REASTO	X
C354	B3CC	REARTS	X
C355	B3CD	OMERR	OUT OF MEMORY ERROR VECTOR
C357	B3CF	ERROR	ERROR HANDLER (ERROR TYPE IN .X)
C364	B3DA	ERRCRD	X
C36A	B3E0	GETERR	X
0000	B3ED	TYPERR	PRINTS OUT THE ERROR MESSAGE
C37E	B3F4	ERRFIN	X
C389	B3FF	READY	PRINTS "READY." GOES INTO MAIN BASIC LOOP (← NMI)
C392	B406	MAIN	MAIN BASIC LOOP, ANALYZES INPUT LINES
C3AB	B41F	MAIN1	LINES THAT START WITH A NUMBER HANDLED HERE
C3E6	B45A	VDECT1	X
C3EE	B462	MLOOP	X
C3FC	B470	NODEL	X
C417	B48B	NODELC	X
C431	B4A5	STOLOP	X
C439	B4AD	FINI	CLEANS BASIC SYSTEM UP; CLR
C442	B4B6	LNKPRG	RELINKS BASIC STATEMENTS IN TEXT AREA
C44B	B4BF	CHEAD	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
C453	B4C7	CZLOOP	X
C46E	B4E1	LNKRTS	X
C46F	B4E2	INLIN	INPUT A LINE OF INFORMATION INTO BUF (MAX 80 CHARS)
C471	B4E4	INLINC	X
C47E	B4F8	FININ1	X
C495	B4FB	CRUNCH	LOOKS UP KEYWORDS IN AN INPUT LINE
C49B	B501	KLOOP	X
C4A7	B50D	CMPSPC	X
C4BD	B523	KLOOP1	X
C4C5	B52B	MUSTCR	X
C4CF	B53D	RESER	X
C4D1	B544	RESCON	X
C4E0	B552	GETBPT	X
C4E2	B554	STUFFH	X
C4F5	B567	COLIS	X
C4F7	B569	NODATT	X
C4FE	B570	STR1	X
C507	B579	STRNG	X
C50E	B580	NTHIS	X
C512	B584	NTHIS1	X
0000	B58D	NTHIS2	X
C522	B599	CRDONE	X
C52C	B5A3	FNDLIN	SEARCHES FOR A LINE NUMBER (NUMBER IN LINNUM)
C530	B5A7	FNDLNC	X
C547	B5BE	FNDLO1	X
C550	B5C7	AFFRTS	X
C559	B5D0	FLINRT	X
C55A	B5D1	FLNRTS	X
C55B	B5D2	SCRATH	IMPLEMENTS "NEW" COMMAND - CLEARS EVERY THING
C55D	B5D4	SCRTCH	X
C572	B5E9	RUNC	X
C577	B5EE	CLEAR	CLR - ROUTINE
C579	B5F0	CLEARC	X
0000	B60B	FLOAD	X
C593	B60E	STKINI	X
C5A6	B621	STKRTS	X
C5A7	B622	STXTPT	TXTPTR=TXTTAB-1
C5B5	B630	LIST	ROUTINE - LIST
C5BD	B638	GOLST	X
C5D4	B64F	LSTEND	X
C5E2	B65D	LIST4	X
C5FF	B67A	TSTDUN	X
C601	B67C	TYPLIN	X
C608	B683	PRIT4	X
C60C	B687	PLOOP	X
C619	B694	PLOOP1	X
C62D	B6A8	GRODY	X
C630	B6AB	QPLOP	X
C642	B6C5	RESRCH	X
C645	B6C8	RESCR1	X
0000	B6CE	RESCR2	X
C64D	B6D4	PRIT3	X
0000	B6D5	PRIT3B	X
C658	B6DE	FOR	ROUTINE - FOR
C669	B6EF	NOTOL	X
C6A1	B727	LDFONE	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
C6B5	B73B	ONEON	X
C6C4	B74A	NEWSTT	MAIN STATEMENT DISPATCH LOOP (DO NEXT STATEMENT)
C6D4	B759	DIRCON	X
C6E4	B769	DIRCN1	X
C6F7	B77C	GONE	DISPATCHES NEXT BYTE CHRGET RETURNS
C700	B785	GONE3	DISPATCHES .A IF NONZERO ELSE LOOP TO NEWSTT
C702	B787	GONE2	X
0000	B795	GONE4	X
C717	B7A2	GLET	X
C71A	B7A5	MORSTS	X
C71E	B7A9	SNERR1	SYNTAX ERROR VECTOR
0000	B7AC	GO	HANDLE GO TOKEN CASE (FIND A TO)
C730	B7B7	RESTOR	ROUTINE - RESTORE
C73A	B7C1	RESFIN	X
C73E	B7C5	ISCRTS	X
C73F	B7C6	STOP	STOP - SEC END - CLC
C741	B7C8	END	ROUTINE - END
C742	B7C9	STOPC	ROUTINE - STOP
C751	B7D8	STPEND	X
C759	B7E0	DIRIS	X
C75B	B7E2	ENDCON	X
C768	B7EB	GORDY	JMP READY
C76B	B7EE	CONT	ROUTINE - CONT
C784	B807	CONTRT	X
C785	B808	RUN	ROUTINE - RUN
C790	B813	GOSUB	ROUTINE - GOSUB
C7A4	B827	RUNC2	X
C7AD	B830	GOTO	ROUTINE - GOTO
C7C4	B847	LUK4IT	X
C7C8	B84B	LUKALL	X
C7D9	B85C	GORTS	X
C7DA	B85D	RETURN	ROUTINE - RETURN
C7EB	B86E	USERR	BAD SUBSCRIPT ERROR VECTOR
C7F0	B873	SNERR2	SYNTAX ERROR VECTORY
C7F3	B876	RETU1	X
C800	B883	DATA	X
C803	B886	ADDON	X
C80D	B890	REMRTS	X
C80E	B891	DATAN	SEARCH FOR NEXT
C811	B894	REMN	LOOK FOR EOL($00) (TXTPTR OFFSET IN .Y)
C819	B89C	EXCHGT	X
C821	B8A4	REMER	X
C830	B8B3	IF	ROUTINE - IF
C83F	B8C2	OKGOTO	X
C843	B8C6	REM	ROUTINE - REM
C848	B8CB	DOCOND	X
C850	B8D3	DOCO	X
C853	B8D6	ONGOTO	ROUTINE - ON (GOTO OR GOSUB)
C85B	B8DE	SNERR3	SYNTAX ERROR VECTOR
C85F	B8E2	ONGLOP	X
C867	B8EA	ONGLP1	X
C872	B8F5	ONGRTS	X
C873	B8F6	LINGET	INPUT A BASIC LINE NUMBER (0-63999)(VALUE IN LINNUM)
C879	B8FC	MORLIN	X
C8A7	B92A	NXTLGC	X
C8AD	B930	LET	ROUTINE - LET

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
C8CA	B94D	QINTGR	X
C8DE	B961	COPFLT	X
C8E1	B964	COPSTR	X
C8E2	B965	INPCOM	X
C8F5	B978	TIMELP	X
C90F	B992	NOML6	X
C91F	B9A2	TIMEST	X
C928	B9AB	TIMNUM	X
C92F	B9B2	FCERR2	ILLEGAL QUANITY ERROR VECTOR
C932	B9B5	GOTNUM	X
C937	B9BA	GETSPT	COPY STRINGS IF NEEDED
0000	B9BE	DSKX0	X
0000	B9D2	DSKX1	X
0000	B9D4	DSKX2	X
C948	B9E1	QVARIA	X
C956	B9EF	DNTCPY	X
C95D	B9F6	COPY	X
C973	BA13	COPYC	X
0000	BA2E	COPY00	X
0000	BA44	COPY01	X
0000	BA46	COPY02	X
0000	BA4E	STRADJ	POINT TO STRING FOR A COPY
0000	BA6C	ADJ	X
0000	BA70	ADJXX	X
0000	BA74	ADJ02	X
0000	BA83	ADJ00	X
0000	BA85	ADJ01	X
C98B	BA88	PRINTN	ROUTINE - PRINT#
C991	BA8E	CMD	ROUTINE - CMD
C99B	BA98	SAVEIT	X
C9A5	BAA2	STRDON	X
C9A8	BAA5	NEWCHR	X
C9AB	BAA8	PRINT	ROUTINE - PRINT
C9AD	BAAA	PRINTC	X
C9D5	BAD2	FININL	X
C9E2	BADF	CRDO	OUTPUT A CARRIAGE RETURN
C9EC	BAED	CRFIN	X
C9EE	BAEF	PRTRTS	X
C9EF	BAF0	COMPRT	X
C9F2	BAF3	MORCO1	X
C9FC	BAFD	TABER	TAB AND SPC HANDLER
CA0C	BB0D	ASPAC	X
CA0D	BB0E	XSPAC	X
CA0E	BB0F	XSPAC2	X
CA11	BB12	NOTABR	X
CA17	BB18	XSPAC1	X
CA1C	BB1D	STROUT	PRINT STRING FROM ADDRESS IN .Y AND .A
CA1F	BB20	STRPRT	PRINT STRING POINTED TO BY INDEX
CA26	BB27	STRPR2	X
CA39	BB3A	OUTSPC	OUTPUT A SPACE
CA40	BB41	CRTSKP	OUTPUT A $1D
CA43	BB44	OUTQST	OUTPUT A ?
CA45	BB46	OUTDO	OUTPUT THE CHAR IN .A
CA4C	BB49	OUTRTS	X
CA4F	BB4C	TRMNOK	HANDLES BAD INPUT DATA
CA59	BB56	GETDTL	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
CA5D	BB5A	STCURL	X
CA61	BB5E	SNERR4	SYNTAX ERROR VECTOR
CA64	BB61	TRMNO1	X
CA6D	BB6A	DOAGIN	X
CA7D	BB7A	GET	ROUTINE - GET OR GET#
CA94	BB91	GETTTY	X
CAA7	BBA4	INPUTN	ROUTINE - INPUT#
CAB7	BBB4	IODONE	RESTORE INPUT TO KEYBOARD
CAB9	BBB6	IORELE	X
CAC1	BBBE	INPUT	ROUTINE - INPUT
CAD2	BBCD	NOTQTI	X
CADA	BBD5	GETAGN	X
CAED	BBE8	BUFFUL	X
0000	BBF1	PTHRTI	X
CAFA	BBF5	QINLIN	PROMPTS AND RECEIVES THE INPUT
CB04	BBFF	GINLIN	X
CB07	BC02	READ	ROUTINE - READ
CB0E	BC09	INPCON	X
CB10	BC0B	INPCO1	X
CB16	BC11	INLOOP	X
CB42	BC3D	QDATA	X
CB4B	BC46	GETNTH	X
CB4E	BC49	DATBK	X
CB52	BC4D	DATBK1	X
CB66	BC61	SETOUT	X
CB72	BC6D	RESETC	X
CB73	BC6E	NOWGET	X
CB7E	BC79	NOWGE1	X
CB8A	BC85	NUMINS	X
CB92	BC8D	STRDN2	X
CB9E	BC99	TRMOK	X
CBB9	BCB4	DATLOP	X
CBD2	BCCD	NOWLIN.	X
CBDF	BCDA	VAREND	X
CBEA	BCE5	VARY0	PRINT "EXTRA IGNORED " IF KEYBOARD AND A SEPERATOR
CBFB	BCF6	INPRTS	X
CBFC	BCF7	EXIGNT	MESSAGE - EXTRA IGNORED
CC0D	BD07	TRYAGN	MESSAGE - ?REDO FROM START
CC20	BD19	NEXT	ROUTINE - NEXT
CC26	BD1F	GETFOR	X
CC29	BD22	STXFOR	X
CC34	BD2D	ERRGO5	X
CC36	BD2F	HAVFOR	X
CC76	BD6F	NEWSGO	X
CC79	BD72	LOOPDN	CHECKS DATA FORMAT
CC8B	BD84	FRMNUM	JMP FRMEVL
CC8E	BD87	CHKNUM	CHECK THAT CURRENT TYPE IS NUMERIC
CC90	BD89	CHKSTR	CHECK THAT CURRENT TYPE IS STRING (CHKS VALTYP)
CC91	BD8A	CHKVAL	X
CC97	BD90	CHKOK	X
CC9A	BD91	DOCSTR	X
CC9A	BD93	CHKERR	TYPE MISMATCH ERROR VECTOR
CC9C	BD95	ERRGO4	X
CC9F	BD98	FRMEVL	FORMULA EVALUATOR - EVALUATES ALL FORMULAS
CCA5	BD9E	FRMEV1	X
CCAA	BDA3	LPOPER	

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
CCB9	BDB2	TSTOP	X
CCBC	BDB5	LOPREL	X
CCD8	BDD1	ENDREL	X
CCF1	BDEA	QPREC	X
CCFA	BDF3	DOPREC	X
CCFB	BDF4	NEGPRC	X
CD08	BE01	FINREL	X
CD12	BE0B	FINRE2	X
CD1A	BE13	QPREC1	X
CD21	BE1A	DOPRE1	PUSHES A PARTIAL EVALUATION ON THE STACK
CD31	BE2A	SNERR5	SYNTAX ERROR VECTOR
CD34	BE2D	PUSHF1	X
CD39	BE32	PUSHF	X
CD44	BE41	FORPSH	X
CD59	BE56	QOP	X
CD5C	BE59	QOPGO	X
CD5E	BE5B	QCHNUM	X
CD65	BE62	UNPSTK	X
CD67	BE64	PULSTK	RESTORE ARG FROM STACK (PUSHED EVALUATION)
CD81	BE7E	QOPRTS	X
CD83	BE80	UNPRTS	X
CD84	BE81	EVAL	EVALUATES NUMERIC FORMULAS
CD88	BE85	EVAL0	X
CD8D	BE8A	EVAL1	X
CD90	BE8D	EVAL2	X
CDA3	BEA0	PIVAL	STORAGE - THE BINARY.VALUE OF PI
CDA8	BEA5	QDOT	X
CDB8	BEB5	STRTXT	IMMEDIATE STRINGS HANDLER
CDC1	BEBE	STRTX2	X
CDC7	BEC4	EVAL3	X
CDCF	BECC	NOTOP	EVAL - NOT
CDDE	BEDB	EVAL4	X
CDEC	BEE9	PARCHK	EVALUATE A FUNCTION WITHIN ()'S (FRMEVL)
CDF2	BEEF	CHKCLS	CHECK FOR RIGHT PARENTHESIS)
CDF5	BEF2	CHKOPN	CHECK FOR LEFT PARENTHESIS (
CDF8	BEF5	CHKCOM	CHECK FOR A COMMA
CDFA	BEF7	SYNCHR	COMPARE TXTPTR AGAINST .A IF THEN...
CE03	BF00	SNERR	...SYNTAX ERROR VECTOR
CE08	BF05	DOMIN	SET UP FUNCTION FOR FUTURE EVALUATION
CE0A	BF07	GONPRC	X
0000	BF0C	CKSMB0	THE CHECKSUM BYTE FOR THE $B000 ROM
0000	BF0D	ISVJMP	JMP ISVAR
0000	BF10	PABB0	PATCHES
0000	BF10	PATCHG	P
0000	BF1D	PCTH0	P
0000	BF1E	PCTH1	P
0000	BF21	PATCHH	P
0000	BF2E	PATCHI	P
CE0F	BF8C	ISVAR	SET UP A VARIABLE NAME SEARCH
CE11	BF8E	ZZ6	X
CE12	BF8F	ISVRET	X
0000	BFC1	ISVDS	DS$ TEST AND HANDLER
CE42	BFD3	STRRTS	X
CE43	BFD4	GOOO	X
CE54	BFE5	GOOOOO	X
0000	BFFC	CHKDS	CHECK FOR A DS VARIABLE

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
CE69	C003	GETTIM	ASSIGN TIME TO TI
CE75	C00F	@STATU	X
0000	C01C	@DSAV	X
CE82	C040	GOMOVF	X
CE89	C047	ISFUN	DISPATCH AND EVAL IF IT'S A FUNCTION
CEB3	C071	OKNORM	X
CEB8	C076	FINGO	PLACE FUNCTIONS DISPATCH ADDRESS IN JUMPER AND GO
CEC8	C086	OROP	EVAL - OR
CECB	C089	ANDOP	EVAL - AND
CEF8	C0B6	DOREL	DO COMPARISONS
CF10	C0CE	STRCMP	X
CF38	C0F6	STASGN	X
CF3D	C0FB	NXTCMP	X
CF43	C101	@COMP	X
CF48	C106	GETCMP	X
CF54	C112	DOCMP	X
CF5D	C11B	GOFLOT	X
CF60	C11E	DIM3	MULTIPLE DIM RE-ENTRY (CHKS FOR A COMMA)
CF63	C121	DIM	ROUTINE - DIM
CF6D	C12B	PTRGET	SEARCHES FOR A BASIC VARIABLE
CF72	C130	PTRGT1	X
CF74	C132	PTRGT2	X
CF7E	C13C	INTERR	SYNTAX ERROR VECTOR
CF81	C13F	PTRGT3	X
CF91	C14F	ISSEC	X
CF92	C150	EATEM	X
CF9C	C15A	NOSEC	X
CFA6	C164	NOTSTR	X
CFB6	C174	TURNON	X
CFBD	C17B	STRNAM	X
CFD3	C18F	STXFND	X
CFD5	C191	LOPFND	X
CFDF	C19B	LOPFN	X
0000	C1AB	NXTPTR	MOVE SEARCH TO NEXT TABLE ENTRY
CFED	C1AC	NOTIT	X
CFF7	C1B6	ISLETC	X
D000	C1BF	ISLRTS	X
D001	C1C0	NOTFNS	DID NOT FIND VARIABLE - CREATE A NEW ONE
D007	C1C6	LDZR	X
D00C	C1CB	NOTEVL	X
D01C	C1DB	GOBADV	X
D01F	C1DE	@STAVR	CHECK FOR ST CASE
0000	C1E6	@DSVAR	CHECK FOR DS CASE
D027	C1F2	VAROK	GOOD USABLE VARIABLE
D03D	C208	NOTEVE	X
D448	C21C	ARYVA2	X
D44C	C220	ARYVA3	X
D457	C228	ARYVGO	SEARCH THE ARRAYS
D488	C259	ARYGET	MOVE THRU THE ARRAY TABLES
D492	C263	GOGO	X
0000	C281	GOGO1	X
D4D0	C290	DVARTS	X
0000	C29D	ARYDON	X
D069	C2B9	FINPTR	LOGS BASIC VARIABLE LOCATION
D073	C2C3	FINNOW	X
D078	C2C8	FMAPTR	ARRAY POINTER SUBROUTINE

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
D084	C2D4	JSRGM	X
D089	C2D9	N32768	STORAGE - THEN BINARY VALUE -32768
D08D	C2DD	INTIDX	EVALUATE FORMULA RESULT IS POSITIVE INTEGER VALUE
D093	C2E3	POSINT	CONVERT FLOATING BINARY TO POSITVE INTEGER
D09A	C2EA	AYINT	CONVERT FLOATING BINARY TO INTEGER
D0A7	C2F7	NONONO	ILLEGAL QUANITY ERROR VECTOR
D0A9	C2F9	QINTGO	JMP QINT
D0AC	C2FC	ISARY	LOCATES AND/OR CREATES ARRAYS
D0B6	C306	INDLOP	X
D0F7	C347	LOPFDA	X
D103	C353	LOPFDV	X
D112	C362	NMARY1	X
D120	C370	BSERR	BAD SUBSCRIPT ERROR VECTOR
D123	C373	FCERR	ILLEGAL QUANITY ERROR VECTOR
D125	C375	ERRGO3	X
D128	C378	GOTARY	X
D13C	C38C	NOTFDD	X
D150	C39F	NOTFLT	X
D159	C3A8	STOMLT	X
D162	C3B1	LOPPTA	X
D172	C3C1	NOTDIM	X
D195	C3E4	GREASE	X
D1A4	C3F3	ZERITA	X
D1A9	C3F8	DECCUR	X
D1C6	C415	GETDEF	X
D1CE	C41D	INLPNM	X
D1E4	C433	BSERR7	SYNTAX ERROR VECTOR
D1E7	C436	OMERR1	OUT OF MEMORY ERROR VECTOR
D1EA	C439	INLPN2	X
D1EB	C43A	INLPN1	X
D1FC	C44B	ADDIND	X
D20D	C45C	NOTFL1	X
D213	C462	STOML1	X
D227	C476	DIMRTS	X
D228	C477	UMULT	INTEGER ARITHMETIC ROUTINES FOR MULTI-DIM ARRAYS
D231	C480	UMULTD	X
D23B	C48A	UMULTC	X
D254	C4A3	UMLCNT	X
D258	C4A7	UMLRTS	X
D259	C4A8	FRE	ROUTINE - FRE(X)
D260	C4AF	NOFREF	X
D26D	C4BC	GIVAYF	CONVERTS INTEGER TO FLOATING BINARY
D27A	C4C9	POS	ROUTINE - POS(X)
D27C	C4CB	SNGFLT	X
D280	C4CF	ERRDIR	IF COMMAND TYPE IS INDIRECT ONLY - ILLEGAL DIRECT
D288	C4D7	ERRGUF	UNDEFINED FUNCTION ERROR VECTOR
D28D	C4DC	DEF	ROUTINE - DEF FN()=
D2BB	C50A	GETFNM	X
D2CE	C51D	FNDOER	EVALUATES FN() IN FORMULAS
D2F2	C541	DEFSTF	X
D329	C578	DEFFIN	X
D33F	C58E	STRD	ROUTINE - STR$
D349	C598	TIMSTR	MAKE A STRING OUT OF INFO AT $01FF
D34F	C59E	STRINI	MAKE A STRING OUT OF (FACMO POINTER)
D357	C5A6	STRSPA	X
D361	C5B0	STRLIT	SCANS AND SETS UP STRING ELEMENTS

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
D367	C5B6	STRLT2	X
D371	C5C0	STRGET	X
D37E	C5CD	STRFIN	X
D382	C5D1	STRFI1	X
D383	C5D2	STRFI2	X
D38F	C5DE	STRST2	X
D399	C5E8	STRCP	X
D3A4	C5F3	PUTNEW	CHECK STRING TEMPS PLACE DATA IN TEMPS
D3AC	C5FB	ERRGO2	X
D3AF	C5FE	PUTNW1	X
D3CE	C61D	GETSPA	BUILDS STRING VECTORS
D3D0	C61F	TRVAG2	X
D3DB	C62D	TRVAG3	X
0000	C63A	TRVAG4	X
D3E5	C644	STRFRE	X
0000	C65A	GETRTS	X
D3F0	C65B	GARBAG	X
D400	C66A	GARBA2	DOES 'GARBAGE COLLECTION' - PACKS STRINGS
0000	C67E	GLOOP	X
0000	C68A	COL00	X
0000	C693	COL00B	X
0000	C69E	COL00A	X
0000	C6A9	COL01	X
0000	C6B2	COL02	X
0000	C6CE	GLOP1	X
0000	C6D8	COL02B	X
0000	C6F0	COL02A	X
0000	C700	GRBEND	JMP ENDGRB
0000	C703	COL03	MOVES FRESPC TO FRETOP
0000	C716	ENDGRB	MOVES FRESPC TO FRETOP
0000	C71F	SKIP2	X
0000	C724	SKIP2A	X
0000	C726	MOVPNT	X
0000	C730	MOV00	X
0000	C735	MOVTOP	X
0000	C73F	MOV01	X
0000	C744	SETINX	X
0000	C746	SET00	X
D517	C74F	CAT	CONCATENATE TWO STRINGS (FAC) AND (++(TXTPTR))
D537	C76F	SIZEOK	X
D554	C78C	MOVINS	X
D562	C79A	MOVSTR	X
D566	C79E	MOVDO	X
D56A	C7A2	MOVLP	X
D573	C7AB	MVDONE	X
D57C	C7B4	MVSTRT	X
D57D	C7B5	FRESTR	X
D580	C7B8	FREFAC	X
D584	C7BC	FRETMP	FREES UP TEMPORARY STRING POINTERS
0000	C7DE	RE300	X
0000	C7F6	FRE01	X
D5AF	C7FC	FREPLA	X
0000	C7FE	FRE02	X
D5B5	C811	FRETMS	X
D5C5	C821	FRERTS	X
D5C6	C822	CHRD	ROUTINE - CHR$(VALUE) (VALUE 0-255)

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
D5DA	C836	LEFTD	ROUTINE - LEFT$()
D5E0	C83C	RLEFT	X
D5E6	C842	RLEFT1	X
D5E7	C843	RLEFT2	X
D5E8	C844	RLEFT3	X
D5FF	C85B	PULMOR	X
D606	C862	RIGHTD	ROUTINE - RIGHT$()
D611	C86D	MIDD	ROUTINE - MID$()
D622	C87E	MID2	X
D63B	C897	PREAM	USED BY RIGHT
D656	C8B2	LEN	ROUTINE - LEN(STRING)
D65C	C8B8	LEN1	X
D665	C8C1	ASC	ROUTINE - ASC(STRING)
D672	C8CE	GOFUC	X
D675	C8D1	GTBYTC	DOES A CHRGET AND GETBYT
D678	C8D4	GETBYT	EVALUATE THE FORMULA AND RETURN A BYTE VALUE (IN .X)
D67B	C8D7	CONINT	X
D687	C8E3	VAL	ROUTINE - VAL(STRING)
D6A7	C903	VAL2	X
D6BD	C918	ST2TXT	X
D6C5	C920	VALRTS	X
D6C6	C921	GETNUM	EVALUATE FORMULA AND RETURN INTEGER VALUE (0-65535)
D6CC	C927	COMBYT	X
D6D2	C92D	GETADR	CONVERT FAC TO VALUE(0-65535) PLACE IN POKER
D6E8	C943	PEEK	ROUTINE - PEEK(X)
D6FB	C94E	GETCON	X
D6FE	C951	DOSGFL	X
D707	C95A	POKE	ROUTINE - POKE X
D710	C963	FNWAIT	ROUTINE - WAIT
D71F	C972	STORDO	X
D723	C976	WAITER	X
D72B	C97E	ZERRTS	X
D72C	C97F	FADDH	ADD 1/2 TO FPB VALUE IN FAC
D733	C986	FSUB	UNPACKS ARGUMENT AND SUBTRACT FPB
D736	C989	FSUBT	FPB SUBRATRACTION ARG-FAC
D76E	C998	FADDS	X
D773	C99D	FADD	UNPACK ARGUMENT INTO ARG DO A FPB ADD
D776	C9A0	FADDT	FPB ADDITION FAC=FAC+ARG
D783	C9AD	FADDC	X
D79F	C9C9	FADDA	X
D7A3	C9CD	FADD1	X
D7AF	C9D9	FADD4	X
D7BB	C9E5	SUBIT	X
D7DE	CA08	FADFLT	X
D7E3	CA0D	NORMAL	NORMALIZE ADDITION AND SUBTRACTION RESULTS
D7E7	CA11	NORM3	X
D803	CA2D	ZEROFC	FAC=0
D805	CA2F	ZEROF1	X
D807	CA31	ZEROML	MAKE SIGN POSITIVE
D80A	CA34	FADD2	X
D829	CA53	NORM2	X
D835	CA5F	NORM1	X
D842	CA6C	SQUEEZ	X
D844	CA6E	RND3HF	X
D852	CA7C	RNDRTS	X
D853	CA7D	NEGFAC	COMPLEMENT FAC ENTIRELY

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
D859	CA83	NEGFCH	COMPLEMENT JUST THE NUMBER IN FAC
D87B	CAA5	INCFAC	INCREMENT FAC
D889	CAB3	INCFRT	X
D88A	CAB4	OVERR	OVERFLOW ERROR VECTOR
D88F	CAB9	MULSHF	SHIFER ROUTINES
D891	CABB	SHFTR2	X
D8A5	CACF	SHIFTR	X
D8B2	CADC	SHFTR3	X
D8B8	CAE2	SHFTR4	X
D8BC	CAE6	ROLSHF	X
D8C6	CAF0	SHFTRT	X
D8C8	CAF2	FONE	FLOATING-POINT-BINARY CONSTANTS
D8CD	CAF7	LOGCN2	X
D8E2	CB0C	SQR05	X
D8E7	CB11	SQR20	X
D8EC	CB16	NEGHLF	X
D8F1	CB1B	LOG2	X
D8F6	CB20	LOG	ROUTINE - LOG(X)
D8FD	CB27	LOGERR	ILLEGAL QUANITY ERROR VECTOR
D900	CB2A	LOG1	X
0000	CB5A	MULLN2	X
D934	CB5E	FMULT	FPB MULTIPLY FAC=FAC*ARG
D937	CB61	FMULTT	FPB MULTIPLY WITH ARG AND .AC LOADED
D965	CB8F	MLTPLY	X
D96A	CB94	MLTPL1	X
D96D	CB97	MLTPL2	X
D989	CBB3	MLTPL3	X
D997	CBC1	MULTRT	X
D998	CBC2	CONUPK	UNPACK MEMORY INTO ARG
D9C3	CBED	MULDIV	CHECK AND ADJUST EXPS OF FPB MULT AND DIV
D9C5	CBEF	MLDEXP	X
D9D0	CBFA	TRYOFF	X
D9E0	CC0A	MLDVEX	X
D9E6	CC10	ZEREMV	X
D9EB	CC15	GOOVER	OVERFLOW ERROR VECTOR
D9EE	CC18	MUL10	MULTIPLY FAC BY 10
D9F9	CC23	FINML6	X
DA04	CC2E	MUL10R	X
DA05	CC2F	TENC	FPB VALUE 10
DA0A	CC34	DIV10	DIVIDE FAC BY 10
DA13	CC3D	FDIVF	X
DA1B	CC45	FDIV	UNPACK MEMORY AND DIVIDE
DA1E	CC48	FDIVT	FAC = ARG/FAC
DA35	CC5F	DIVIDE	X
DA4B	CC75	SAVQUO	X
DA58	CC82	QSHFT	X
DA5B	CC85	SHFARG	X
DA69	CC93	DIVSUB	X
DA86	CCB0	LD100	X
DA8A	CCB4	DIVNRM	X
DA96	CCC0	DVOERR	OVERFLOW ERROR VECTOR
DA9B	CCC5	MOVFR	MOVE RES TO FAC
DAAE	CCD8	MOVFM	MOVE MEMORY TO FAC
DAD3	CCFD	MOV2F	X
DAD6	CD00	MOV1F	X
DADC	CD06	MOVVF	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
DAE0	CD0A	MOVMF	MOVE FAC TO MEMORY
DB08	CD32	MOVFA	MOVE ARG TO FAC
DB0A	CD34	MOVFA1	X
DB0E	CD38	MOVFAL	X
DB18	CD42	MOVAF	MOVE FAC TO ARG
DB1B	CD45	MOVEF	X
DB1D	CD47	MOVAFL	X
DB26	CD50	MOVRTS	X
DB27	CD51	ROUND	ROUND FAC
DB2F	CD59	INCRND	X
DB37	CD61	SIGN	EXTRACT SIGN FROM FAC IN .A
DB3B	CD65	FCSIGN	X
DB3D	CD67	FCOMPS	X
DB44	CD6E	SIGNRT	X
DB45	CD6F	SGN	ROUTINE - SGN(X)
DB48	CD72	FLOAT	FLOAT THE SIGNED INTEGER IN FAC
DB50	CD7A	FLOATS	FLOAT THE SIGNED NUMBER IN FAC
DB55	CD7F	FLOATC	X
DB5B	CD85	FLOATB	X
DB64	CD8E	ABS	ROUTINE - ABS(X)
DB67	CD91	FCOMP	COMPARE ARG AND FAC .A=1←A<F
DB69	CD93	FCOMPN	X
DB9E	CDC8	FCOMPC	X
DBA4	CDCE	FCOMPD	X
DBA7	CDD1	@INT	FAC=INT(FAC) SIGNED ROUTINE - INT(X)
DBBB	CDE5	@ISHFT	X
DBC6	CDF0	@INTRT	X
DBC7	CDF1	@INT1	X
DBD8	CE02	INT	ROUTINE - INT(X)
DBF5	CE1F	CLRFAC	.A TO ALL POSISTIONS OF FAC
DBFE	CE28	INTRTS	X
DBFF	CE29	FIN	FBP INPUT, TXTPTR POINTS TO ASCII, RETURNS IN FAC
DC03	CE2D	FINZLP	X
DC12	CE3C	@PLUS	X
DC16	CE40	FINC	X
DC19	CE43	FINDG@	X
DC1B	CE45	FIN1	X
DC3A	CE64	FINEC1	X
DC3C	CE66	FINEC	X
DC3F	CE69	FNEDG1	X
DC41	CE6B	FINEC2	X
DC4D	CE77	FINDP	X
DC53	CE7D	FINE	X
DC55	CE7F	FINE1	X
DC5E	CE88	FINDIV	X
DC67	CE91	FINMUL	X.
DC6E	CE98	FIN@NG	X
DC73	CE9D	NEGX@S	X
DC76	CEA0	FINDIG	X
DC7D	CEA7	FINDG1	X
DC8A	CEB4	FINLOG	X
DC9D	CEC7	FINEDG	X
DCAC	CED6	MLEX10	X
DCBA	CEE4	MLEXMI	X
DCBF	CEE9	N@999	FPB VALUE 99999999.90625
DCC4	CEEE	N9999	FPB VALUE 999999999.5

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
DCC9	CEF3	NMIL	FPB VALUE 10-9
0000	CEF8	CKSMC0	CHECKSUM BYTE $C000 ROM
DCCE	CF78	INPRT	PRINT CURRENT LINE NUMBER
DCD9	CF83	LINPRT	PRINT NUMBER IN (.A←HIGH .Y←LOW)
DCE6	CF90	STROU2	JMP STROUT
DCE9	CF93	FOUT	FPB OUTPUT
DCEB	CF95	FOUTC	X
DCF3	CF9D	FOUT1	X
DD0C	CFB6	FOUT37	X
DD15	CFBF	FOUT7	X
DD17	CFC1	FOUT4	X
DD22	CFCC	FOUT3	X
DD2D	CFD7	FOUT38	X
DD34	CFDE	FOUT9	X
DD3B	CFE5	FOUT5	X
DD3E	CFE8	BIGGES	X
DD53	CFFD	FOUTPI	X
DD54	CFFE	FOUT6	X
DD5F	D009	FOUT39	X
DD70	D01A	FOUT16	X
DD72	D01C	FOUT8	X
DD74	D01E	FOUTIM	CLOCK ENTRY INTO FOUT
DD76	D020	FOUT2	X
DD9A	D044	FOUT41	X
DD9C	D046	FOUT40	X
DDA3	D04D	FOUTYP	X
DDBE	D068	STXBUF	X
DDD0	D07A	FOULDY	X
DDD2	D07C	FOUT11	X
DDDF	D089	FOUT12	X
DDEF	D099	FOUT14	X
DDFB	D0A5	FOUT15	X
DE10	D0BA	FOUT19	X
DE13	D0BD	FOUT17	X
DE18	D0C2	FOUT20	X
DE1D	D0C7	FHALF	FPB VALUE 1/2
DE1F	D0C9	ZERO	X
DE22	D0CC	FOUTBL	TABLES OF POWERS OF -10↑X
DE46	D0F0	FDCEND	END OF POWERS TABLE
DE5E	D108	TIMEND	FPB TIME CONVERSION TABLES
DE5E	D108	SQR	ROUTINE - SQR(X)
DE68	D112	FPWRT	ROUTINE (ARG↑FAC)
DE71	D11B	FPWRT1	X
DE8B	D135	FPWR1	X
DEA1	D14B	NEGOP	NEGATE THE NUMBER IN FAC
DEAB	D155	NEGRTS	X
DEAC	D156	LOGEB2	FPB VALUE LOG(E) BASE 2
DEB1	D15B	EXPCON	LOG AND EXPONENT FPB TABLES
DEDA	D184	EXP	ROUTINE - EXP(FAC)
DEEA	D194	STOLD	X
DEF5	D19F	GOMLDV	X
DEF8	D1A2	EXP1	X
DF08	D1B2	SWAPLP.	X
DF2D	D1D7	POLYX	POLYNOMIAL EVALUATOR
DF43	D1ED	POLY	POLYNOMIAL EVALUATOR
DF47	D1F1	POLY1	X

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
DF56	D200	POLY3	X
DF5A	D204	POLY2	X
DF67	D211	POLY4	X
DF77	D221	RMULC	X
DF7B	D225	RADDC	X
DF7F	D229	RND	ROUTINE - RND(X)
DF9D	D247	QSETNR	X
DFB2	D25C	RND1	X
DFC2	D26C	STRNEX	X
DFD5	D27F	GMOVMF	X
DFD8	D282	COS	ROTINE - COS(X)
DFDF	D289	SIN	ROUTINE - SIN(FAC)
E011	D2BB	SIN1	X
E014	D2BE	SIN2	X
E021	D2CB	SIN3	X
E028	D2D2	TAN	ROUTINE - TAN(FAC)
E050	D2FA	COSC	X
E054	D2FE	PI2	FPB VALUE PI/2
E059	D303	TWOPI	FBP VALUE 2*PI
E05E	D308	FR4	FPB VALUE 1/4
E063	D30D	SINCON	SIN TABLES FPB VALUES
E08C	D32C	ATN	ROUTINE - ATN(FAC)
E094	D334	ATN1	X
E0A2	D342	ATN2	X
E0B5	D355	ATN3	X
E0BB	D35B	ATN4	X
E0BC	D35C	ATNCON	X
E0F9	D399	INITAT	BASIC SYSTEM INITIALIZATION CODE
E0FF	D39F	CHDGOT	X
E110	D3B0	CHDRTS	X
0000	D3B6	INIT	BASIC SYSTEM INITIALIZATION ROUTINE
E131	D3C9	MOVCHG	X
E15D	D400	LOOPMM	X
E165	D408	LOOPM1	X
E174	D417	USEDEC	X
E178	D41B	USEDEF	X
E1B7	D44B	WORDS	MESSAGE - 'BYTES FREE'
E1C4	D458	FREMES	MESSAGE - '### COMMODORE BASIC ###'
E1DE	D472	LASTWR	LAST BYTE OF BASIC SYSTEM CODE+1
0000	DEA4	PATCH2	PATCHES
E844	E844	CHTIM	X
0000	FF93	CONCAT	VECTOR - CONCAT
0000	FF96	DOPEN	VECTOR - DOPEN
0000	FF99	DCLOSE	VECTOR - DCLOSE
0000	FF9C	RECORD	VECTOR - RECORD
0000	FF9F	FORMAT	VECTOR - FORMAT
0000	FFA2	COLECT	VECTOR - COLLECT
0000	FFA5	BACKUP	VECTOR - BACKUP
0000	FFA8	DCOPY	VECTOR - COPY
0000	FFAB	APPEND	VECTOR - APPEND
0000	FFAE	DSAVE	VECTOR - DSAVE
0000	FFB1	DLOAD	VECTOR - DLOAD
0000	FFB4	DIRCAT	VECTOR - DIRECTORY
0000	FFB4	DCAT	VECTOR - CATALOG
0000	FFB7	RENAME	VECTOR - RENAME
0000	FFBA	SCRATC	VECTOR - SCRATCH

Tabella D-3. Indirizzi in esadecimale e etichette di riferimento per BASIC CBM (continua)

BASIC 3.0	BASIC 4.0	Etichette	Descrizione
0000	FFBD	READDS	VECTOR - DS AND DS$
FFC0	FFC0	COPEN	VECTOR - OPEN
FFC3	FFC3	CCLOS	VECTOR - CLOSE
FFC6	FFC6	CDIN	VECTOR - SET INPUT DEVICE
FFC9	FFC9	CDOUT	VECTOR - SET OUTPUT DEVICE
FFCC	FFCC	CLSCHN	VECTOR - RESTORE NORMAL I/O DEVICES
FFCC	FFCC	CCCHN	SAME AS ABOVE
C481	FFCF	INCHR	VECTOR - INPUT A CHARACTER (FROM SCREEN)
FFCF	FFCF	CINCH	SAME AS ABOVE
FFD2	FFD2	OUTCH	VECTOR - OUTPUT A CHARACTER
FFD5	FFD5	CLOAD	VECTOR - LOAD
FFD8	FFD8	CSAVE	VECTOR - SAVE
FFDB	FFDB	CVERF	VECTOR - VERIFY
FFDE	FFDE	CSYS	VECTOR - SYS
FFE1	FFE1	ISCNTC	VECTOR - TEST STOP KEY
FFE4	FFE4	CGETL	VECTOR - GET CHARACTER FROM KEYBOARD BUFFER
FFE7	FFE7	CCALL	VECTOR - ABORT ALL I/O CHANNELS
000F	0000	CONTW	Z
000D	0000	CNTWFL	Z
000F	0000	LINWID	Z
0010	0000	NCMWID	Z
006C	0000	STRNGI	Z
007F	0000	0	Z
C494	0000	INCRTS	Z
C721	0000	SNERRX	Z
D404	0000	FNDVAR	Z
D41E	0000	TVAR	Z
D427	0000	SVARS	Z
D433	0000	SVAR	Z
D43B	0000	SVARGO	Z
D440	0000	ARYVAR	Z
D48A	0000	ARYSTR	Z
D497	0000	DVARS	Z
D4A1	0000	DVAR.	Z
D4B6	0000	DVAR2	Z
D4C0	0000	DVAR3	Z
D4DB	0000	GRBRTS	Z
D4E0	0000	GRBPAS	Z
D5B0	0000	FRETRT	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z

INDICE ANALITICO

ABS 144, 433
APPEND 398
Array vedere Variabili con indici
ASC 433
ASCII, codice 450, 451
Assembler 385
ATN 145, 433
BACKUP 398
BELL 445
BLOCK ALLOCATE 392
BLOCK EXECUTE 391
BLOCK READ 390
BLOCK WRITE 391
Buffer 149, 255
BUFFER POINTER 392
Cassette (file) 261
Cassette (nastri) 65
CATALOG (o DIRECTORY) 311
CHR$ 434
CLOSE 398
CLR 399
CMD 399
COLLECT 400
CONCAT 400
Concatenamento 153, 329
CONT 401
Copie
  di file su dischetto 308
  di programmi 340
COPY 401, 421
COS 145, 434
Costanti 379
CURSOR LEFT 14
CURSOR RIGHT 15
DATA 402
DCLOSE 402
DEF FN 403
DELETE 84
DELETE LINE 445
Demagnetizzatore 29
DIM 404

DIRECTORY 404
Dischetti 70
Dischetti (file) 294
Disk Operating System (DOS) 70, 301
DLOAD 405
DOPEN 405
DS, DS$ 434
DSAVE 406
Editor 83, 445
END 406
ERASE BEGIN 446
ERASE END 446
ESCAPE 18
Esecuzione di programmi
  in modo differito, a programma 60
  in modo immediato 51, 93
EXP 144, 435
File ad accesso diretto (Random) 389
File relativi 331
File sequenziali 318
FOR ... NEXT 407
FRE 436
Funzioni 143, 433
GET 408
GET # 337, 408
GOSUB 409
GOTO 409
Grafica 226
GRAPHIC 446
HEADER 410
IEEE 488 6
IF ... THEN 410
INITIALIZE 422
INPUT 411
INPUT # 412
INSERT 86
INSERT LINE 447
Insieme di caratteri
  alternativo 64
  standard 64
INT 144, 436
Interprete BASIC CBM 379
Istruzioni 122, 398

LEFT$ 436
LEN 437
Linguaggi
  Assembler 385
  BASIC 98
LIST 413
LOAD 414
LOAD & RUN 78
LOG 145, 437
Messaggi di errore 455
MDI$ 437
Modo differito (o a programma) 60
Modo immediato 51, 93
NEW 416, 423
ON ... GOSUB 416
ON ... GOTO 417
OPEN 417
Operatori 108
  aritmetici 108
  Booleani 113
  relazionali 112
Orologio (clock) 235
  "attimi" (jiffy) 236
Parole riservate 119
PEEK 438
POKE 419
Porte utente parallele 6
POS 438
PRINT 419
PRINT # 420
Protezione da scrittura 33, 41
RANDOM 242
READ 426
RECORD # 426
REM 427
RENAME 424, 427
RESTORE 428
RETURN 428
REVERSE ON/OFF 12
RIGHT$ 439
RND 145, 439
RON revisione livello 2 477
RUN 428

SAVE 429
SCRATCH 424, 430
SCROLL DOWN 447
SCROLL UP 447
SET BOTTOM 448
SET TOP 448
SGN 144, 440
SIN 145, 440
Sintassi 98, 459
Sistema Operativo a Disco (DOS) 70, 301
SPC 440
SQR 144, 441
ST 441
Stampanti 80, 342
STOP 431
STR$ 441
Stringhe 104, 379
Subroutine (Sottoprogrammi) 132, 409, 442
SYS 442
TAB 442
TAN 145, 443
Tasti 16
TEXT 448
TI, TI$ 444
USR 444
VAL 444
VALIDATE 425
Variabili 105
Variabili con indici (array) 116, 381
Variabili numeriche 57, 376
VERIFY 431
Virgolette, edizione di un testo tra 87
WAIT 432

Questo è il secondo volume della versione italiana del famosissimo testo americano "PET/CBM Personal Computer Guide".
In questo manuale trovate tutto ciò che è necessario sapere sulla manutenzione dei Personal Computer, sulla memorizzazione di programmi su cassetta o su dischetto e come gestire gli archivi di dati, i "file", sia sequenziali che ad accesso diretto.
Il linguaggio BASIC CBM è presentato in maniera facile mediante tanti esempi di programmi.
Anche il lettore più "digiuno" di calcolatori può trovare, nei due volumi di questo manuale, il modo per avvicinarsi al fantastico ed entusiasmante mondo dei personal computer.

L. 22.500

Cod. 333 P ISBN 88-7056-030-9

97

PET/CBM GUIDA ALL'USO VOL. 2

Adam Osborne
Carrol S. Donahue

GRUPPO EDITORIALE JACKSON